GUÍA RÁPIDA #01

DESARROLLO DE VIDEOJUEGOS PARA SINCLAIR *ZX-SPECTRUM*

Edición 1.0b

[en desarrollo...]

19/01/2019

https://calentamientoglobalacelerado.net/zxopensource

RAFAEL LOMEÑA VARO

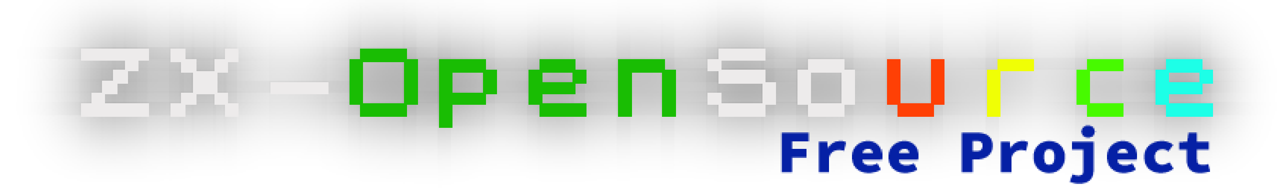
MaRaF SOFT ©© MMXIX





Guía Rápida de Desarrollo basada en la versión `0.37b` del juego tipo *El Chatarrero Galáctico*

00
Indice temático



00
Prólogo del autor

Hace muuUUUuucho tiempo, algunos jóvenes adolescentes tuvieron un sueño que no pudieron hacer realidad, hoy, treinta y cinco años después, tal vez ese sueño pueda resucitar gracias a la *retroescena* y a humildes proyectos como *ZXOpenSource*[1].

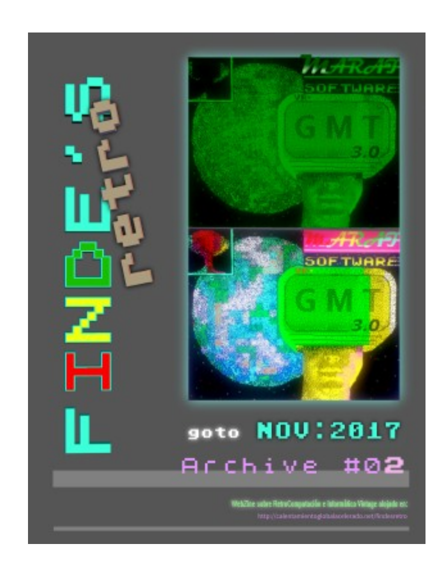
A todos aquellos jóvenes entusiastas que un día persiguieron con pasión la felicidad y la encontraron frente a una máquina llamada *Spectrum*, está dedicada esta *Guía Rápida de Desarrollo*, a todos ellos y también, a esos jóvenes intrépidos que quieran aventurarse a vivir hoy en primera persona lo que un día fue el origen de la mayor industria de la historia.

Por ello, y por el considerable a la vez que reconfortante esfuerzo que ha supuesto para mí escribir estar humilde obra, solo espero que puedas encontrar en ella una puerta experimental de entrada al mundo del desarrollo *homebrew* para plataformas de *8 bits*.

1 **ZXOpenSource** *es un proyecto abierto iniciado en 2018con el juego El Chatarrero Galáctico que pretende ser una puerta experimental de entrada a desarrollo hombrew para ZX-Spectrum. Website oficial:* https://calentamientoglobalacelerado.net/zxopensource

01
Génesis del Proyecto

La extravagante idea de crear una guía rápida para el desarrollo de videojuegos destinada a la plataforma de 8 bits *ZX-Spectrum* surgió, como otras tantas casualidades de la vida, a raíz de un sencillo programa cuyo código escribí sobre mi flamante *Spectrum +3* una amena tarde de domingo y de la que también surgió el *webzine* *FINDE's RETRO*[2].



Casualidades de la vida que a veces nos sorprenden. Es el azar en su estado puro, fruto y a la vez origen de la transcendencia vital que nos rodea. Mi *Spectrum* +3 lo recuperé de un centro de tratamiento de residuos antes de que fuera destruido y mi fantástica TV de 14' la recogí junto a un contendor de basura cuando pasé de milagro por una calle de mi ciudad. Ahora que lo pienso, tal vez, toda esta

2 **FINDES RETRO** *es un magazine digital y disponible online en formato web 1.0 sobre contenido diverso relacionado con la retrocomputación y publicado en la url* https://calentamientoglobalacelerado.net/findesretro

chatarra electrónica tan valiosa para unos y tan superflua para otros, tuvo alguna influencia subliminar a la hora de escoger la temática del juego al que estaba a punto de dedicarle los próximos meses de mi vida y que posteriormente iban a dar origen a esta propia guía y al proyecto *ZX-OPENSOURCE*[3].

Lo cierto, es que andaba yo trasteando estos viejos cacharros con esa pasión característica propia de cualquier retro-nostálgico que se precie cuando, al mover la palanca de un joystick *Sinclair* conectado a mi flamante +3 (yo nunca tuve uno para jugar con mi gomas al *Túneles Marcianos* y por eso me mataban), me resultó curioso ver como se mostraban secuencias de números sobre la pantalla dependiendo de la dirección en la que movía la misma ... ¡¡Uaaaaaauu!! En ese preciso instante comenzó a volar mi imaginación exactamente igual que cuando, siendo casi un niño, me sentaba a programar frente a mi gomas y el televisor de tubo de 14', y cuando me quise dar cuenta estaba picando código como un poseso disfrutando de mi flamante +3... ¡Y volví a sentir mariposas en mi estómago!!!

En un pequeño espacio de tiempo, aunque esta es la primera noción que suele perderse cuando la

3 **ZX-OPENSOURCE:** *ambicioso proyecto abierto iniciado con el fin de divulgar el conocimiento y las herramientas necesarias para el apoyo de autores al desarrollo hombrew de la plataforma 8-bits ZX-Spectrum. La presente guía forma parte del propio proyecto ZX-OpenSource, nacido sobre la filosofía de código abierto. La web oficial del projecto ZX es accesible en la url:* https://calentamientoglobalacelerado.net/zxopensource

programación te absorbe, ya tenía el esqueleto de un gracioso jueguecillo de 1 *Kbyte* que hacía uso de la palanca de juegos y al que decidí bautizar con el rimbombante nombre de *EL CHATARRERO GALÁCTICO.*

Ahora, será el *EL CHATARRERO GALÁCTICO* el puente que nos llevará hasta el conocimiento práctico que, junto a dosis necesarias de imaginación, te permitirán adentrarte en el apasionante campo que ya se ha dado en llamar la *escena retro* o *homebrew*[4].



4 **Homebrew** *es el término inglés internacionalizado y utilizado para referirse a la escena retro o retroescena (retroscene) que enmarca el desarrollo de software para antiguas plataformas informáticas de 8 y 16 bits comercializadas principalmente durante las décadas de 1980 y 1990.*

02
Requisitos de Conocimientos Previos

Probablemente, para iniciar el desarrollo de un juego relativamente serio para el microordenador *ZX-Spectrum* habrás comenzado preguntándote: **¿Puedo realmente programar un juego de calidad aceptable sin ser un mago del código máquina?** La respuesta a esta pregunta es **SÍ**, en mayúsculas, pero es importante que conozcas ciertos matices.

El lenguaje de programación *BASIC* del *Sinclair ZX-Spectrum* y al que simplemente llamaremos ***ZXBASIC*** en lo sucesivo, adolece de ciertas limitaciones que unas veces podrás sortear con algo de ingenio y otras con el uso de ***compiladores***[5] de código como ***HIBASIC*** ó ***MCODER.*** Los compiladores de código pueden imprimir mucha más velocidad a tu juego, con todas las ventajas que ello conlleva, abriendo una nueva dimensión para el desarrollador ajeno al código máquina, pero debes saber que programar en ***ZXBASIC*** implica ciertas

5 **COMPILADORES:** *Los compiladores de código son programas que convierten el código escrito en ZXBASIC a código máquina que, al ejecutarse directamente y de forma no interpretada (como en el caso del ZXBASIC) muestran una velocidad de ejecución muy superior con respecto al código original en ZXBASIC interpretado. Estas mejoras pueden variar bastante de un compilador a otro y también depende de las funciones utilizadas, pero pueden llegar a suponer en ocasiones específicas incrementos en el rendimiento de hasta un 400%*

limitaciones infranquables que nunca podrás superar ni aún con el mejor de los compiladores. Por tanto, debes ser consciente desde el principio que estas limitaciones condicionarán el diseño de tus juegos si vas a trabajar en ***ZXBASIC compilado***.

Con respecto a los compiladores existen varias opciones y aunque yo personalmente vengo utilizando la versión 3 de ***MCODER***, existen otras herramientas incluso algo más flexibles como ***HISOFT***. Si queremos usar herramientas más potentes y modernas, también existen otras alternativas interesantes como son los compiladores cruzados para plataformas *Windows* ó *LiNUX*, de forma que, si un día decides desarrollar juegos para ***ZX-Spectrum*** utilizando otro lenguaje de programación distinto al ***ZXBASIC***, puedes explorar nuevos y potentes lenguajes compilados como el ***BASIC BORIEL*** o el uso de la ***librería Z88*** para compiladores ***C*** y que genera código máquina compatible con el procesador ***Z80A***[6] del *Spectrum* (muy recomendable el *Curso Z88DK del gurú RADASTAN* accesible en ***speccy.org***), o incluso, por qué no, valorar el aprendizaje del propio ensamblador para ***Z80*** que además te servirá para conocer a fondo otras plataformas de *8 bits* y quién sabe si para aprobar alguna que otra asignatura si eres estudiante de informática. La programación en

6 **Z80A** *es la variedad del microprocesador de 8 bits Z80 con una frecuencia mayor que el Z80 original (3.58 Mhz en el caso del Spectrum frente a 2.5 Mhz de las primeras versiones) lanzado por la compañía Zilog en julio de 1976 y que montaban los microordenadores ZX-Spectrum. Existen versiones que alcanzan los 20 Mhz.*

código máquina podrás hacerla sobre la máquina original (con el uso de un ensamblador nativo o incluso ensamblado a mano con un listado de mnemónicos) o bien mediante el uso de compiladores cruzados sobre *Windows* o *LiNuX* que generan código máquina para ***Z80***, como ***PASMO***.

En nuestro caso, el lenguaje que vamos a utilizar en todo momento será el *BASIC* nativo del *Sinclair ZX-Spectrum* al que, como ya te comenté, de ahora en adelante denominaremos ***ZXBASIC***, y aunque sería absurdo incluir toda la documentación relativa a este lenguaje y sus comandos dado que ese conocimiento ya se encuentra recogido de forma magistral en numerosas publicaciones técnicas y revistas especializadas de la época, sí te recomiendo tener a mano el manual oficial del usuario de cualquier ordenador *Sinclair ZX-Spectrum*, en el que encontrarás todos los comandos disponibles de este lenguaje y la forma de utilizarlos. En este sentido, quiero dejar constancia de que la detallada lectura del manual original que acompañaba a los microordenadores ***ZX-Spectrum*** suele ser una buena base para adentrarse en el desarrollo, sobre todo si partes de cero. En la red existen numerosos repositorios en los que se hallan preservados todos estos libros y documentos de forma digital, normalmente en formato PDF y puedes encontrar una completa lista de estos almacenes/repositorios

en la dirección web del proyecto ***GUTENBERG 3.0***[7] o descargar directamente el manual del *ZX-Spectrum* desde el *Taller de Descarga de Recursos* que encontrarás al final de esta guía.

Por otro lado, aunque puede que no sea estrictamente necesario que domines ningún lenguaje de programación antes de entrar en materia, seguramente será más fácil para ti si ya has programado algo previamente, si sabes lo que es una variable y un bucle, por ejemplo, o si tienes algunas nociones de programación en cualquier otro lenguaje, aunque sean mínimas.

Aún así, si no has programado nunca pronto descubrirás que la programación es siempre un reto trepidante y especialmente enriquecedor para desarrollar la creatividad. Sólo tú podrás darte cuenta si te merece la pena pasar horas creando un juego por puro placer. Es así de sencillo, disfrutar por encima de todo.

7 **Gutenberg 3.0** *es un proyecto personal y completamente abierto destinado a la gestión de bibliotecas locales y en unidades de red que resuelve el problema del indexado masivo tanto en entornos domésticos como corporativos. El proyecto es actualmente accesible en la dirección web https://calentamientoglobalacelerado.net/gutenberg30. En su primer recopilatorio experimental contaba con más de 1000 manuales digitalizados en formato PDF relativas a computadoras clásicas de la segunda mitad del siglo XX.*

03
Objetivos de la Guía Rápida

La presente *Guía Rápida* no es ni un curso de programación en lenguaje *BASIC,* ni mucho menos un tutorial de *Ensamblador/Código Máquina*[8]. Esta guía está basada en la creación y análisis de un sencillo juego arcade tipo cuyo código, escrito en ***ZXBASIC,*** estudiaremos a fondo y nos servirá de soporte para conocer las técnicas más básicas a las que deberás enfrentarte para abordar cualquier desarrollo de estas características.

Cómo podrás comprobar a medida que vayas avanzado en la guía, no se trata de un folleto en el que consultar la sintaxis de un lenguaje o una lista detallada de comandos, sino de un documento en el que se analizan conceptos diferentes a los de un mero manual técnico o de programación. Por otro lado, resulta obvio que, dado su volumen, será necesario un tiempo y dedicación para asimilar su contenido. Esto te llevará tu tiempo, tal vez unas

8 **ENSAMBLADOR / Código Máquina** *es el nombre con el que se denomina al lenguaje de programación de bajo nivel ligado directamente al procesador. Si bien ENSAMBLADOR es el nombre que se le da al lenguaje en sí, aunque por asociación también se le conoce sencillamente como CÓDIGO MÁQUINA. Frente a la altísima velocidad de ejecución de sus programas y el control total del hardware se encuentra la enorme dificultad de programación que supone y el profundo conocimiento del microprocesador que se requiere para ello.*

semanas a fondo o algo más si partes de cero pero siempre dependerá del entusiasmo con el que te lo tomes y de los conocimientos previos que poseas, y en cualquier caso, con el conocimiento y experiencia que adquirirás al estudiar en profundidad esta *guía rápida* espero que puedas conseguir:

- Escribir sencillos juegos con un acabado y calidad suficiente para conseguir que sean difundidos a través de La Red y reconocidos, por ejemplo, en concursos de desarrollo *hombrew* e indizados en los repositorios oficiales online de software para *ZX-Spectrum*.

- Aprender más sobre la mítica plataforma *ZX-Spectrum*, saboreando la intensa y nada desdeñable satisfacción personal de crear un juego por ti mismo, o incluso, por qué no ...

- ¡¡Hacer historia con un juego espectacular que sea elegido *GOTY (game of the year)* 2025 en el grandioso portal www.elmundodelspectrum.com !! ¡El ingenio no tiene límites y tu *ZX-Spectrum* tampoco!

Para lograr estos objetivos, nuestra *guía rápida* nos aportará de forma detallada:

- Una visión global de nuestras posibilidades de desarrollo conociendo las limitaciones del *BASIC* de *Sinclair (ZXBASIC)* para el desarrollo de videojuegos así como el

soporte y las ventajas ofrecidas por el compilador de código ***ZXBASIC*** nativo ***MCODER3***.

- Comprensión y uso de los comandos más frecuentes del lenguaje ***ZXBASIC*** necesarios para el desarrollo de sencillos videojuegos, ***LET, PRINT, PLOT, DRAW, INK, PAPER, INKEY$, GOTO, BEEP, FOR...NEXT***, etc.

- Comprensión y uso de las variables típicas para el desarrollo de un juego, destinadas principalmente al control de coordenadas y contadores, así como la interpretación y uso de éstas mediante la lógica condicional que nos ofrece el lenguaje.

- Distribución de la pantalla del *ZX-Spectrum* en 2 dimensiones (ejes X, Y), tanto en alta como en baja resolución (*Ver anexos*).

- Técnicas sencillas de control y movimiento de objetos/personajes en pantalla mediante el teclado o la palanca de juegos/Joystick.

- Técnicas posibles para la detección de objetos en pantalla para lograr la **interacción funcional** del personaje con el escenario y posibles colisiones con objetos.

- Elaboración y uso de los sprites/*GDUs*

mediante herramientas específicas de apoyo al diseño y gestión de gráficos tales como *ZX-DRAW* ó *GDUCalc*[9].

- Refresco de variables durante el juego (Fuel, Score, coordenadas, etc.) y visualización en pantalla de barras de progreso (combustible, tiempo, etc) en tiempo real .

- Técnica básica utilizada para la introducción de música *background* (de fondo) con intercepción de teclado.

- Depuración de errores y pulido del aspecto visual del programa al máximo mediante la continua revisión de código.

- Última revisión del código para optimización máxima y mejora del rendimiento, especialmente del motor principal del juego. Llevar el código al límite con una revisión exhaustiva sobre todo de los bucles.

- Compilación y ensamblado del código fuente en *ZXBASIC* mediante el compilador nativo ***MCODER3***. Algo con lo que muchos

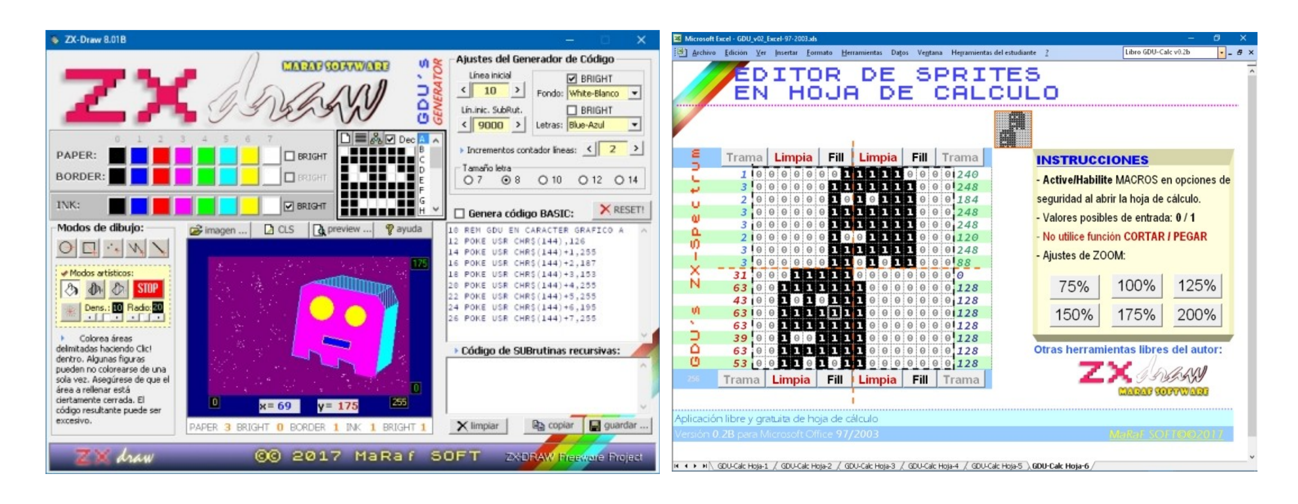
9 ***ZX-DRAW*** *es una suerte de editor gráfico cruzado (para sistemas Windows) que yo mismo programé para el diseño de pantallas utilizables en el ordenador ZX-Spectrum de Sinclair mediante generación automática de código BASIC-ZX. Se trata en realidad de una aplicación de ingeniería inversa ya que el código se va generando a partir del diseño que hacemos en pantalla.* ***GDUCalc****, a diferencia de ZX-Draw, es un aplicativo basado en hoja de cálculo y dirigido específicamente al desarrollo y gestión de bibliotecas de GDU's que también genera código ZX-BASIC. (Ver capturas de ambas aplicaciones en la página siguiente)*

soñamos un día lejano pero nunca llegó … ¡hasta hoy!

- Elaboración de pantalla de carga para versión de cinta física o en formato de cinta virtual (archivos de extensión *TAP*) mediante herramientas cruzadas (*ZX-Draw*, etc.) o nativas (*Artist*, etc.)

- Elaboración y montaje final del juego en versiones digitales y/o físicas y su publicación/venta/intercambio en portales especializados o retroeventos.

La mayor parte de estos conocimientos serán adquiridos de forma práctica, deshilachando línea a línea el código completo de nuestro juego *EL CHATARRERO GALÁCTICO* y estudiando a fondo la funcionalidad de cada orden a través de los comentarios de las mismas en los que iremos viendo su forma de uso y finalidad.



04
Como Enfocar Nuestro Proyecto

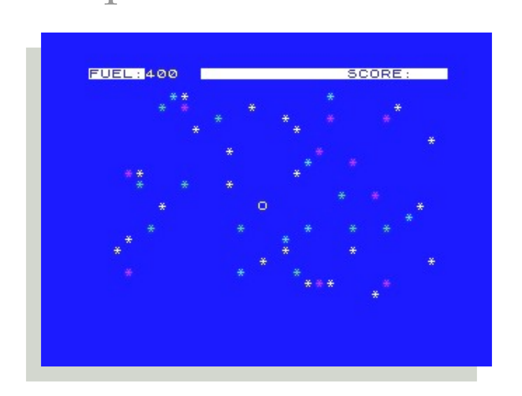
A la hora de abordar el desarrollo de un juego, lo primero que necesitamos es la semilla. Con semilla me refiero a una idea más o menos clara de lo que deberá ser nuestro juego una vez acabado, o al menos, por dónde vamos a comenzar nuestro trabajo. Dibujarlo en nuestra mente y tomar notas de todo cuanto se nos ocurra es siempre buena idea para dar comienzo a un desarrollo.

En esta fase, tan creativa como trepidante, es fundamental tener consciencia de las limitaciones a las que el ***harwdare*** y el ***software***[10] nos someterá para no acabar dando al traste con nuestros proyectos. Podemos soñar con juegos maravillosos de personajes fantásticos que viven apasionadas aventuras en bellos paisajes eléctricos pero lo mejor es comenzar por algo sencillo como nuestro proyecto *EL CHATARRERO GALÁCTICO*. A veces, una idea primitiva y sencilla puede ser la forma más adecuada de comenzar a conocer una plataforma y

10 **HARDWARE y SOFTWARE:** *ambos conceptos representan la dualidad en la esencia del ordenador, el cuerpo y el alma de la computadora que conforman cualquier sistema informático. El hardware es referido a la parte física que conforma la máquina (circuitos, baterías, procesadores, etc.) mientras que el concepto de software abarca toda la parte inmaterial del sistema tales como archivos y programas.*

conseguir acabar un desarrollo con ciertas garantías de éxito.

De acuerdo con esta línea, hay que reconocer que la idea sobre *El Chaterrero Galáctico* no puede ser más simple. No en vano, en su primera versión se limitaba a una letra *O* que circulaba por la pantalla tragándose asteriscos y un contador de objetos recogidos. Después, hay que trabajar en enriquecer la idea con una historia, menús, gráficos, sonidos y cualquier cosa que se nos ocurra para mejorar nuestro juego funcional y visualmente hasta conseguir darle un atractivo, una personalidad propia que sea capaz de despertar el interés de algún usuario que no sea el propio padre de la criatura. Por ejemplo, en nuestro juego convertiremos la pantalla en una ventana del espacio sideral y los objetos en residuos eléctronicos que gravitan a nuestro alrededor.



Por otro lado, y aunque el detalle sea más técnico que otra cosa, el movimiento de nuestro personaje se hará carácter a carácter ya que el movimiento píxel a píxel queda reservado, casi en exclusiva, para la programación en código máquina. No obstante, no debemos obviar que algunos títulos comerciales de gran éxito en su momento tampoco gozaron en su

día de un fluido movimiento píxel a píxel. Para comprender la diferencia entre ambos tipos de movimientos puedes ojear la distribución de pantalla en ambos modos (alta y baja resolución) en la ***sección de anexos*** que encontrarás al final de la presente guía.

En este aspecto, otro detalle a tener en cuenta es la ventaja que ofrece el ***Spectrum*** al poder utilizar de forma simultánea los modos de baja y alta resolución, que pueden coexistir sin problemas. Esto, aunque pueda parecer trivial, no ocurría en otras plataformas como ***ORIC***, en el que la visualización de objetos en modo de *alta resolución* no era compatible con el modo de *baja resolución* o *modo texto*.

Para estructurar bien el juego, puede ser buena idea establecer de alguna forma diferentes niveles de dificultad, por ejemplo, en nuestro juego tipo, dado que contamos con una cantidad de combustible limitada, se podrá aumentar la dificultad en función del número de objetos a recoger, con ello, ya tenemos un criterio para establecer diferentes niveles.

Además de estas cuestiones, existen ciertas fórmulas para imprimir mayor dinamismo a nuestro juego, como por ejemplo:

- Representar al protagonista con una imagen dinámica que se vaya alternando mientras está parado.

- Incluir un sutil efecto de sonido al recoger cada objeto o residuo electrónico que, por ejemplo, vaya incrementando el tono a medida que vayan quedando menos objetos por recoger, aportando así algo más de tensión al ambiente.

- Experimentar bastante con los colores con la idea de conseguir una ambientación adecuada y vistosa. ¡Los bellos colores de nuestro *Spectrum* están ahí para ser utilizados!

Por otro lado y resumiendo ya el guión básico de nuestro juego, el protagonista (el Chatarrero) irá recorriendo la pantalla (convertida en sectores del espacio sideral) recogiendo los objetos (residuos de chatarra electrónica) y superando los niveles establecidos al limpiar completamente cada sector.

Nuestra idea está lista para empezar a ser implementada, así que reune todas tus notas y bocetos y siéntate ante la computadora. ¡¡Es hora de picar código!! pero asegúrate antes de que podrás salvar tu trabajo de forma segura al final de cada jornada. Si trabajas con un emulador no tendrás problema para ello, pero si lo haces en un *Spectrum* real, haz algunas pruebas antes y medita muy bien

el soporte de memoria que utilizarás en tus copias, pues no hay nada más frustrante para un programador que la pérdida accidental de su trabajo.

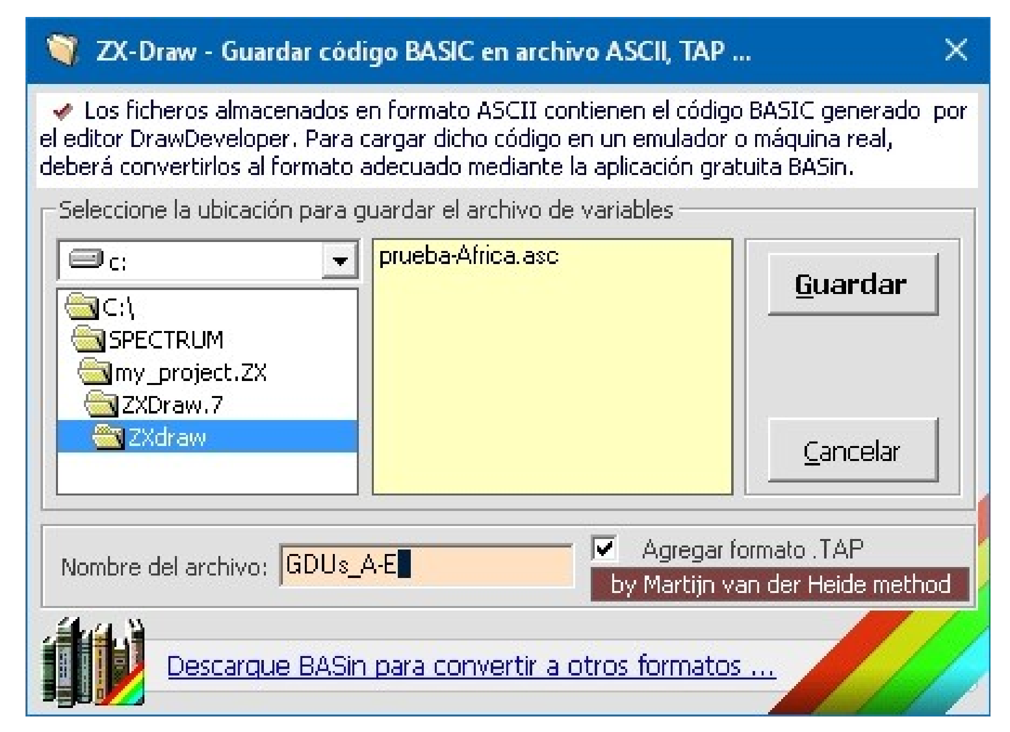


Ilustración 1: *Si bien las herramientas actuales ofrecen infinidad de posibilidades para migrar códigos de un formato a otro, conviene tener muy claro cuales son nuestras expectativas y las herramientas que utilizaremos para alcanzarlas. En la imagen se muestra la ventana de guardado de código ZXBASIC generado por ZX-DRAW en archivos de texto ASCII, reutilizables a su vez desde otros entornos como BASinC.*

05
Las Vías Técnicas de Desarrollo

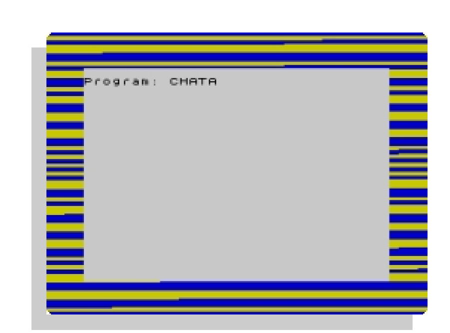
Una vez clara la idea de lo que queremos hacer nos toca decidirnos entre diversas vías técnicas para crear nuestro juego y que a veces vendrá determinada en función de las características del propio desarrollo (tipo de juego) y/o de nuestras propias preferencias de trabajo (lenguajes o herramientas preferidas principalmente).

Dado el colapso informativo al que a veces nos somete *la Red*, voy a intentar resumir, grosso modo, las distintas alternativas posibles en un desarrollo de las características del ***Chatarrero Galáctico*** son:

1. **ARCAICA o purista:** La antigua usanza está recomendada especialmente para pequeños desarrollos, gente con mucho tiempo libre u "ortodoxos radicales" (sin intención de ofensa;). Básicamente consiste en el desarrollo



mediante el uso de la plataforma original (*gomas*, +2, +3, etc.) y con todos los encantos y limitaciones que esto supone. Dependiendo del lenguaje o herramienta software que se utilice, dentro de esta primera



opción se puede optar también por diversas *subvías* de desarrollo. Por ejemplo, si no queremos meternos con ***ZXBASIC*** ó ***Código Máquina***, podemos recurrir a software nativo de la *época dorada* como ***Game Designer*** *I/II (Software Studios ©1983),* el cual nos permite crear sencillos videojuegos de tipo arcade a partir de modelos predefinidos. En el caso concreto de *Game Designer*, se trata de un diseñador de juegos dotado de un motor en código máquina y un robusto interface que nos permitirá producir sencillos juegos arcades tipo (marcianitos, tanques, etc.) con calidad comercial aceptable (genera juegos muy optimizados) y con tiempos de desarrollo cortos al no escribir una sola línea de código, pero por otro lado, bastante encorsetados en cuanto a la estructura o guión del desarrollo. Podemos decir que este tipo software entronca directamente con las herramientas de autor que permiten al usuario la creación de ciertas aplicaciones sin necesidad de escribir un solo comando. El resultado, aun otorgando cierta libertad creativa al autor, suele arrojar desarrollos

de patrones muy similares y fácilmente identificables.

2. **INTERMEDIA o equilibrada:** Esta opción es la recomendada para seguir nuestra guía rápida y apuesta por el desarrollo "casi directo" sobre la plataforma *ZX-Spectrum* pero corriendo en un emulador como ***Spectaculator*** o el poderoso entorno cruzado de desarrollo ***BASINc***. Con el emulador ***Spectaculator*** editamos el código de forma "nativa" pero trabajando sobre nuestro sistema *Windows*® y, ya puestos a pedir, lo más cómodo es seleccionar desde el emulador cualquier modelo de ***Spectrum 128***[11] aunque nuestro juego final se vaya a compilar para máquinas de 48K. Para ello solo debemos seleccionar en el menú de arranque el ***128 BASIC*** ó ***+3 BASIC,*** aunque cada vez que queramos compilar el código con ***MCODER3*** será necesario conmutar desde el ***MODO 128*** al ***MODO 48*** tecleando el comando ***SPECTRUM***. El único fin de trabajar en el ***MODO 128*** es poder teclear las órdenes de forma natural a diferencia del ***MODO***



11 **SPECTRUM 128:** *Estas máquinas consituyen una auténtica segunda generación de Spectrum y permiten operar en dos modos diferentes de trabajo, el* ***modo 128*** *y, para mantener la retrocompatibilidad, el anterior* ***modo 48K****. Para ello, disponen de una suerte de "sistema operativo" que nos permite seleccionar los distintos modos de trabajo y/o otras funciones. Lo realmente interesante es que, aunque vayamos a compilar nuestro programa para correr en ordenadores de 48K, siempre podemos escribir nuestro código desde el modo 128K y aprovecharnos de la mayor versatilidad y comodidad del editor del 128. Básicamente, se comercializaron tres modelos de 128: el plus 1 ó simplemente + (primera versión); +2 (casette incorporado); y +3 (incorpora disco de 3" en la consola).*

48 que solo permite la entrada de órdenes mediante las teclas asignadas por la *ROM (tokens del Spectrum 48),* además de aprovecharnos de la potente función ***RENUM*** que nos permite reordenar todas las líneas de código de nuestro programa y actualizando automáticamente los saltos de los comandos ***GOTO*** y ***GOSUB***. En esta vía intermedia de desarrollo también podemos editar nuestro código con el potente aplicativo ***BASINc*** y combinar esta versátil herramienta con el uso del emulador ***Spectaculator*** (o cualquier otro), siempre que al guardar nuestro programa en un fichero utilicemos para ello un formato compatible, como puede ser el archipopular ***.SNA.*** Al optar por esta vía de desarrollo, podemos disfrutar la potencia infinita y el tacto exquisito de emuladores como *Spectaculator, ZEsarUX, FUSE, ZXSpin, Retro Virtual Machine* y otros muchos disponibles hoy día para prácticamente cualquier plataforma, o como he dicho anteriormente, del mágico entorno de desarrollo ***BASINc*** para *Windows* que nos permitirá trabajar en modo casi *SPECTRUM* sobre un editor cruzado con todas las comodidades a las que ya estamos acostumbrados, copiar, cortar, pegar, etc. así como de otras muchas utilidades accesorias que veremos con mayor detalle en ***esta sección de***



herramientas de apoyo. Entiendo que es ésta sin duda (la vía intermedia) una opción bastante atractiva y equilibrada para los que se sienten cómodos y gustan saborear del viejo lenguaje ***ZXBASIC*** trabajando con la máquina original (aunque sea emulada) pero sin renunciar a muchos privilegios exclusivos no disponibles en nuestra querida plataforma de *8 bits*. Algunas de las ventajas a las que me refiero son principalmente la carga y el salvado inmediato de programas, velocidad de frecuencia del procesador ***Z80A*** configurable (muy interesante para compilar el código a velocidad de la luz;), ajustes de visualización con filtros de imagen mejorados, edición automatizada de ***GDUs***, emulación de todo tipo imaginable de periféricos y un larguísimo e inacabable etc.

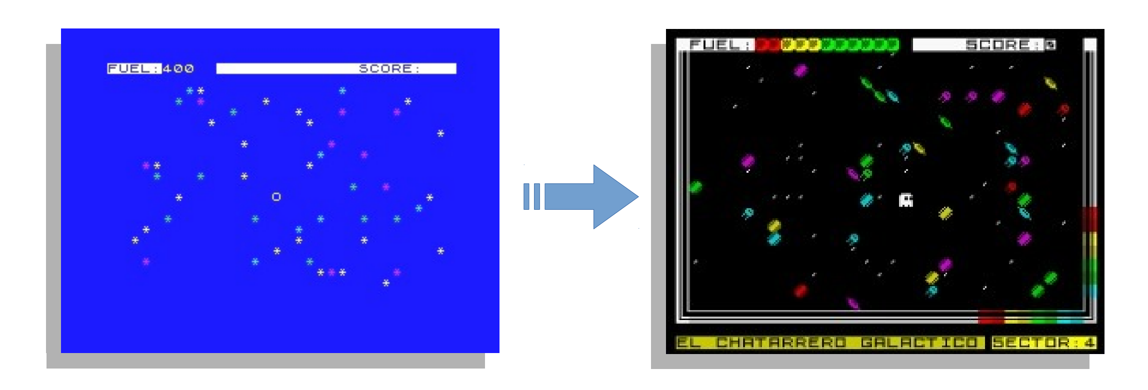
3. **VANGUARDISTA o revolucionaria:** Recomendada para gurús dispuestos a llevar el hardware de *8 bits* al límite y hacer historia, opta por escoger el uso de modernas herramientas cruzadas específicas para el desarrollo de videojuegos y que corren sobre plataformas *PC (Windows/LiNuX)*, tales como *La Churrera (de los Mojons Twins - MK1/MK2) ó AGD (de Jonathan Cauldwell)*. La potencia de algunas de estas herramientas nos permiten llevar a cabo desarrollos de calidad comercial muy alta sin necesidad de escribir todo el código de nuestro

proyecto desde cero y prácticamente sin limitaciones. De hecho, una gran parte de juegos desarrollados en la actualidad están siendo creados con estas potentes herramientas y entre éstos se incluyen géneros tan diversos que van desde sencillos arcades clásicos hasta espectaculares isométricos 3D, un espectro al que solo la imaginación puede poner límites. Sin embargo, la potencia de estas herramientas de desarrollo está ligada a su complejidad y exprimirlas al máximo puede requerir de un conocimiento exhaustivo de las mismas, más allá del mero conocimiento del ***ZXBASIC***. Además, como te comenté al inicio de esta guía, si un día decides desarrollar juegos para ***ZX-Spectrum*** utilizando otro lenguaje de programación distinto al ***ZXBASIC*** y contar con herramientas más versátiles y mayores posibilidades, puedes explorar nuevos y potentes lenguajes compilados como el poderoso ***BASIC BORIEL*** o el uso de la ***librería Z88*** para compiladores ***C*** (muy recomendable el *Curso Z88DK de RADASTAN* en ***speccy.org***) que generan un código máquina ejecutable compatible con el procesador ***Z80A***[12] del *Spectrum* muy optimizado. Por otro lado, la programación directa en código máquina, aunque para la inmensa mayoría de los mortales puede que resulte demasiada complicada como para

12 **Z80A** *es la variedad del microprocesador de 8 bits Z80 con una frecuencia mayor que el Z80 original (3.58 Mhz en el caso del Spectrum frente a 2.5 Mhz de las primeras versiones) lanzado por la compañía Zilog en julio de 1976 y que montaban los microordenadores ZX-Spectrum. Existen versiones que alcanzan los 20 Mhz.*

abordar un proyecto completo en solitario, también podrás llevarla tanto sobre la máquina original (con el uso de un ensamblador nativo o compilando a mano con un listado de mnemónicos y el propio ***ZXBASIC*** como cargador de código) como a través de compiladores cruzados para *Windows* ó *LiNuX* que generan código máquina para el procesador ***Z80***, ***PASMO*** es el más conocido.

En cualquier caso y sea cual sea la vía por la que nos decidamos, de lo que se trata al fin y al cabo es de modelar nuestra idea e ir dándole forma poco a poco hasta pasar de algo muy sencillo y casi espartano a un producto final algo más refinado y atractivo, y aunque supongo que en trabajos de más envergadura las fases del desarrollo no se correspondan con las aquí descritas, este ha sido el ciclo de vida y la evolución del juego que hoy nos ocupa.



Para lograr este objetivo no queda otro camino que ir añadiendo, probando y depurando código que

mejore nuestro programa no solo centrados en el aspecto visual, sino también en lo que afecta al guión o la funcionalidad del juego, ya sabes, dificultad, niveles, jugabilidad y cualquier otra cosa que se nos ocurra que consideremos que puede ampliar su atractivo.

Estas notas son, a modo de borrador, algunas de las que yo describí durante el desarrollo de ***EL CHATARRERO GALÁCTICO*** y tal vez puedan darte una idea bastante aproximada de lo que pretendo explicarte:

- *Se crea menú principal de juego que dará acceso a los distintos niveles de dificultad.*

- *Se implementan GDUs (elaborados con ZX-Draw) de caracterización del personaje y objetos (chatarra). Para ello se guardan los diseños en archivo con formato* ***.TAP*** *y se* ***MERGE****a luego con el código principal del proyecto (ojo con que no coincidan los números de línea) o bien se pega (probar) el código generado directamente sobre el programa usando el entorno* ***BASINc****.*

- *Se dibuja al chatarrero en alta resolución (con ZX-Draw) y se muestra en el menú principal. Después modificar atributos de zonas de pantalla que afectan al dibujo mediante*

*comando **POKE dir.screen,atributo** y se encierra en bucle principal del menú para producir efecto destellante y continuo.*

- *Revisar código e incluir comentarios aclaratorios para la fase de desarrollo.*

- *Se mejora el sonido simulado del motor de la nave del chatarrero (usando RND) y se implementan sprites dinámicos para nuestra nave que varíen en función de la dirección y en posicionamiento estático/inmovil del prota.*

- *Implementar sencilla rutina que sitúa aleatoriamente cuerpos celestes simulando el espacio profundo. Optimizar.*

- *Se incorpora barra de estado que muestra nombre juego y nivel en curso.*

- *Se implementa código que evita el borrado de las estrellas cuando pasa el sprite del chatarrero sobre ellas.*

- *Se implementa melodía final del juego al superar el nivel, introduciendo comandos de interrupciones de teclado entre la secuencia de BEEP's para poder salir de la música al pulsar una tecla.*

- *Se implementa efecto sonoro in crescendo*

para la recogida de chatarra que aumenta el tono a medida que va limpiando el sector (persigue aumentar tensión del jugador ;)

- *Revisión general de parámetros y asociaciones de color.*

- *Se modifica el juego de caracteres (tipo de letra) por defecto mediante minúscula rutina en código máquina que se carga al inicio del programa. ¡Ojo! no ejecutar en modo 128 pues cuelga. Anulada durante el desarrollo con REM, se activa solo en MODO 48 antes de compilar.*

- *Se realiza compilado de todo el código fuente con MCODER 3.*

- *Tras compilar el código, se guarda todo el programa resultante en un archivo .TAP mediante la orden SAVE "NOMB-PRG" y generando el fichero .TAP (en SPECTACULATOR hay que activar/mostrar la grabadora y poner un fichero vacío con cualquier nombre y extensión .TAP) en el interior de la grabadora virtual (se puede arrastrar y soltar dentro).*

- *Las versiones intermedias del desarrollo siempre pueden ir guardándose en **SNA** con nombres de archivo y número incremental de versión (ej. **CHATAv037.SNA**).*

06
Hardware y Software Necesario

Considerando que hemos optado por la ***segunda vía*** del apartado anterior o ***vía intermedia***, el único hardware que necesitas para comenzar a seguir la guía o empezar tu proyecto es un ordenador *compatible PC* con un sistema operativo *Windows de 32 bits* (***XP*** como mínimo aunque otros más antiguos también podrían funcionar). Aunque cueste de creer, incluso un viejo *PC* con procesador ***iPentium*** ó ***AMD K6*** podría convertirse en una potente estación de trabajo y desarrollo que colmaría sin duda nuestras expectativas. Por otro lado, si trabajas en *LiNuX* tampoco tendrás problemas para encontrar las herramientas necesarias, sobre todo un buen emulador. Aunque *LiNuX* pueda ofrecer un repertorio de emuladores inferior al sistema de *Microsoft*, he oído que ***FUSE*** es el mejor emulador de *ZX-Spectrum* para los sistemas del pingüino, por supuesto, con permiso del activo y veterano ***ZEsarUX*** *(by César Hernández)* y el recién llegado a la fiesta por la puerta grande, ***Retro Virtual Machine*** *(by Juan Carlos González Amestoy)* que además de correr de forma nativa en plataformas linuxeras

tambíen están disponibles para *MacOS*.

Como verás, ni siquiera necesitas la máquina original para la que vamos a desarrollar, y esto es algo que debemos agradecer eternamente a los magos y programadores que dedican su valioso tiempo a la creación de esos programas mágicos llamados ***emuladores***, ellos son los auténticos protagonistas de toda esta historia y a ellos le debemos verdaderamente todo lo que la escena retro ha llegado y pueda llegar a ser.

Por otro lado, todas las herramientas de software que he utilizado para el desarrollo del juego modelo de ***El Chatarrero Galáctico*** así como las herramientas y utilidades (gratuitas) que aquí se citan, puedes descargarlas en cualquier momento desde el *taller de descarga* que encontrarás al final de la presente guía. Enumerémoslas:

- Emulador ***SPECTACOL v5.3*** para *Windows*®: última versión libre de este estupendo y versátil emulador de *Spectrum* para sistemas operativos *Windows* y disponible también para otros sistemas operativos. No obstante, puede que necesitemos en algún momento usar otro emulador para tareas específicas como por ejemplo, imprimir nuestro código en una impresora virtual *ZX-Printer* o, grabar una versión ***.TAP*** de nuestro juego, etc. Como pronto descubrirás, esto no representa un gran problema

dados los muchos y maravillosos emuladores que tenemos para elegir. Mi consejo es que si vamos a desarrollar cualquier tipo de aplicaciones para nuestro querido *Spectrum* es interesante que creemos una carpeta con un buen montón de emuladores para utilizar en cualquier momento. Aunque puedan incorporar algún instalador, la mayoría de ellos son portables lo cual siempre es de agradecer si somos de los que nos gusta mantener el registro de Windows lo más ligero posible.

- Compilador ***MCODER III*** para *ZX-Spectrum*: ***MCODER*** es un potente compilador nativo de código ***ZXBASIC*** que otorgará más potencia y versatilidad a tu desarrollo y lo que es mejor, es sencillo, muy sencillo. La versión 3 de este compilador parece que mejoró entre otras cosas el tratamiento de matrices de datos y, aunque conviene no abusar del uso de variables dimensionales (arrays), los pocos problemas que he tenido de compilación han podido solucionarse considerando algunos pequeños detalles. En este sentido, es importante que vayamos compilando nuestro proyecto de forma periódica y/o cada vez que añadamos algo de código, por ejemplo, al final de cada jornada de trabajo, utilizando para guardar el nuevo fichero un número correlativo para cada compilación y así mantener las anteriores versiones. Esta será siempre una buena costumbre para que podamos localizar y depurar más fácilmente

posibles incompatibilidades de nuestro código con el compilador ***MCODER3*** (o cualquier otro). Estos ***compilados sucesivos***[13] son el ridículo precio a pagar por disfrutar de un compilado seguro garantizar que jamás perderemos el control de nuestro proyecto. algo que te aseguro merece la pena y mucho. Aunque el compilador no genera en ningún momento un código máquina tan optimizado como podría conseguirse mediante la programación a bajo nivel en ensamblador, algunas funciones volarán literalmente ante nuestros ojos llegando a multiplicar la velocidad de ejecución por varios enteros. De hecho, algunos juegos y aplicaciones de la época comercial del *Spectrum* eran en realidad programas en *ZXBASIC* directamente compilados para su comercialización, y algunos de ellos llegaron incluso a consagrarse como clásicos que tal vez recuerdes como *Saimazoon*, *Babaliba* o la popular aplicación de diseño gráfico *Artist.*

13 **COMPILADOS SUCESIVOS:** *El compilado sucesivo en diferentes ficheros con nombres secuenciales del tipo JUEGO_v0.XX, siempre nos permitirá mantener todas las versiones anteriores de nuestro programa y también será especialmente útil a la hora de detectar y depurar a tiempo posibles errores de compilación. Además, la compilación con MCODER3 desde cualquier emulador puede acelerarse fácilmente a nuestro antojo multiplicando la velocidad de nuestro Spectrum emulado, por lo que las compilaciones serán rapidísimas, lejos de los tiempos de compilado que tendríamos que esperar en un Spectrum real.*

07
Otras Herramientas Cruzadas de Apoyo

Cuando hace 35 años alguien con la intención de escribir un programa se sentaba frente a una computadora conectada a un magnetófono y a un televisor que mostraba un mensaje del tipo "*© 1982 Sinclair Research LTD*" o similar, era bastante consciente del desafío al que se enfrentaba y de cuales eran sus límites.

Por suerte o por desgracia, las cosas han cambiado bastante en el ámbito del desarrollo informático. Incluso si nos decidimos por la sofisticada vía "*Vanguardista o Revolucionaria*" (citada en la sección *5 - Vías Técnicas de Desarrollo*) para romper las barreras del ***ZXBASIC*** y por ejemplo, escribir nuestros juegos en ***C*** haciendo uso del kit de librerías ***Z88***, muy pronto percibiremos como el uso de librerías mágicas nos permitirán obtener resultados brillantes con poco esfuerzo.

Esto es bueno en parte, ya que nos permite centrar nuestros esfuerzos en otras tareas más artísticas o trascendentales además de poder conseguir

resultados más brillantes y profesionales, pero por otro lado abre un debate profundo en ciertos ámbitos del desarrollo porque, como si de una caja negra se tratase, nuestro programa acaba ejecutando procesos que no comprendemos ni podemos controlar.

En nuestra decisión por la "*Vía Intermedia y Equilibrada*", optamos sin embargo por el uso de herramientas que pueden facilitarnos tareas lentas y tediosas pero sin perder la esencia básica del desafío al que me refería al inicio de esta sección. Podemos compilar a la velocidad de la luz nuestro programa, abrir 4 ventanas simultáneas emulando un 16K, un 48K, un +2 y un +3, copiar y pegar código de un lado a otro o añadir código a nuestro juego para el sintetizador de voz *Currah*, pero nunca podremos usar una librería mágica para hacer por ejemplo un suave *SCROLL* de pantalla... ¿O sí? Bueno, lo cierto es que el uso de rutinas en ensamblador desde ***ZXBASIC*** es un tema apasionante que nos permite romper muchos límites del lenguaje *ZXBASIC* (interpretado o compilado) y aunque como ya comentamos al inicio de esta guía que el código máquina no será estudiado, sí nos reservamos el derecho de aderezar nuestro juego con una diminuta rutina en código máquina que perfila levemente el tipo de letra de nuestro *Spectrum*, otorgando a nuestra creación un acabado algo más pulido.

Continuando con las posibles herramientas de apoyo, encontramos algunas aplicaciones muy útiles para el desarrollo en general y que conviene explorar en profundidad, tales como:

- Aplicación ***BASINc v1.69***: Potente y completísimo entorno de desarrollo para *ZX-Spectrum* derivado del proyecto *BASin* escrito en *Delphi* para sistemas *Windows* de 32 bits. ***BASINc*** es sin duda la joya de la corona. Probablemente, se trate del entorno de edición y desarrollo más cómodo y potente que existe. Se trata de un entorno completo que integra no sólo un editor de código con todas las facilidades propias de los modernos editores, sino que nos permite ir probando el código en un emulador que se despliega en la ejecución y ver así el resultado de nuestro código cada vez que lo necesitemos. Además del editor de código, *BASINc* incorpora multitud de funciones extras a explorar que nos permitirán renumerar las líneas de nuestros programas, editar nuestros GDUs, guardar nuestros programas en diversos formatos, etc.

- ***ZX-DRAW*** es un sencillo software cruzado (corre sobre cualquier plataforma *Windows de 32 y 64 bits*) pensado para ayudarnos en el diseño gráfico de pantallas en alta resolución. Se trata de un desarrollo propio que se encuentra actualmente en fase beta avanzada pero completamente funcional

y de libre distribución. *ZX-Draw* es capaz de generar código *ZXBASIC* a partir del diseño que vayamos creando en la pantalla virtual. Este código generado de forma automática por la aplicación puede reutilizarse en nuestros programas. La filosofía de esta utilidad está inspirada en la vieja técnica de dibujo mediante papel carbón (calca) e incorpora algunas funciones especiales como aerógrafo, relleno de áreas, etc. La última versión disponible (*v0.9b*) incorpora un sencillo generador de *GDUs* que ha sido utilizado en nuestro programa *El Chatarrero Galáctico* y que nos facilita la engorrosa tarea de crear *sprites GDUs*.

Esta herramienta posiblemente corra sin problemas en cualquier sistema *LiNuX* mediante la utilidad *WINE* que permite ejecutar software *Windows* sobre plataformas *LiNuX*.

Si desea más información acerca de esta herramienta de apoyo puede visitar la web del programa en: https://calentamientoglobalacelerado.net/zxdraw

- **GDUCalc** es un software cruzado basado en hoja de cálculo, por lo que, si bien corre sobre cualquier plataforma *Windows, LiNuX ó MacOS,* necesitará tener instalado *Microsoft EXCEL o Libre CALC* en estos sistemas. Ha sido diseñado específicamente

para una eficaz y cómoda gestión de bibliotecas de sprites GDUs para el ordenador *ZX-Spectrum*. Al descargar esta aplicación encontrarás varias hojas de cálculo idénticas que son en realidad versiones para ***Microsoft Office*** 2003/2007, ***LibreOffice*** v4+ y ***OpenOffice*** v4+.

Si desea más información acerca de esta herramienta de apoyo puede visitar la web del programa en: https://calentamientoglobalacelerado.net/gducalc

Aunque prácticamente no necesitamos mucho más, si lo que deseamos es montarnos un entorno "profesional" de desarrollo para *ZX-Spectrum* de auténtica vanguardia, no está de más que instalemos y experimentemos con algunas otras herramientas de apoyo que sin duda encontraremos en la Red a poco que hagamos una búsqueda precisa. Recuerda que en nuestra *vía técnica intermedia* trabajar en un desarrollo para nuestro entrañable *ZX-Spectrum* no tiene por qué significar darle la espalda a la realidad de los tiempos que corren y a las muchas ventajas que la tecnología informática pone a nuestro alcance. Tal vez por ello, algunos de los últimos desarrollos lanzados en la actualidad para plataformas retro, ya sea en ***ZXBASIC*** o en código máquina, comienzan a

acercarse e incluso a superar a las obras magistrales de la época. Debemos entender que desarrollar nuestro proyecto, aún programando desde cero y sin servirnos de herramientas específicas para la creación de videojuegos tales como las ***MK*** ó ***AGD***, no significa que debamos renunciar al uso de tecnologías y aplicaciones accesorias de apoyo que faciliten nuestro trabajo y que nos permitan mejorar ampliamente la calidad de nuestro producto.

08
Primer Vistazo Superficial al Código

Este apartado nos servirá como calentamiento para ir estructurando nuestra mente. Digamos que nos prepara para asimilar mejor el contenido de las secciones siguientes. Sin entrar de pleno en la profundidad ni los detalles de la programación, haremos primero un recorrido más ligero sobre el programa sobrevolando desde las alturas y examinando todo a vista de pájaro, intentando tomar una visión panorámica que nos permita comprender el funcionamiento de todo lo que hace el programa y el esquema general de éste.

Para ello, he dividido el código completo en bloques que iremos desgranando poco a poco con el único objeto de asimilar mejor el *análisis pormenorizado* al que nos dedicaremos más adelante.

Es lógico que en cualquier momento puedas volver sobre esta sección para releer algunas de sus explicaciones y resfrecar las ideas. ¡Adelante pues!!

```
   1 REM **** EL CHATARRERO GALA
CTICO ****
   2 PAPER 0: BORDER 0: BRIGHT 0
: CLS : RANDOMIZE : GO SUB 1000:
 REM carga SPRITEs: A=█ B=█ C=█
D=█ E=█ F=█ G=█ H=◢ P=█ Q=█ R=█
S=¡
   3 GO TO 2000: REM FONTs- solo
 en modo 48K
   4 CLS : GO SUB 900: GO SUB 13
05: GO SUB 1400: REM PARA REINIC
IAR JUEGO GOTO 4
   5 PRINT #1; INK T+2; BRIGHT 0
; INVERSE 1;"EL CHATARRERO GALAC
TICO █"; BRIGHT 1;"SECTOR:";t
   6 INK 7: BRIGHT 1: FOR n=1 TO
 45: PRINT AT INT (RND*19)+1,INT
 (RND*29)+1;"´": NEXT n
   7 LET r=0: LET cb=t*5: BRIGHT
 0: REM contador de basura; nume
ro de basuras=nivel t*10
   8 FOR n=1 TO cb: LET x=RND*29
+1: LET y=RND*19+1: IF ATTR (y,x
)>6 THEN PRINT AT y,x; INK INT (
n/5)+1;"█": LET r=r+1
   9 NEXT n
  10 FOR n=1 TO cb: LET x=RND*29
+1: LET y=RND*19+1: IF ATTR (y,x
)>6 THEN PRINT AT y,x; INK INT (
n/5)+1;"█": LET r=r+1
  11 NEXT n
  12 PRINT AT 0,0; INK 7; BRIGHT
 1; INVERSE 1;" FUEL:"; INVERSE
0;"    "; INVERSE 1;"
 SCORE:"; INVERSE 0;"0   "; INVER
SE 1;" "
  13 LET t=t*10: LET s=0: LET f=
150: LET x=15: LET y=12: LET g$=
"█": LET d$=" ": PRINT BRIGHT 1;
AT 0,6; INK 2;"██"; INK 6;"███";
 INK 4;"██████ "
  14 INK 7: BRIGHT 1: REM color
nave cambia a 6 al comer
```

```
  15 IF INKEY$="6" AND x>1 THEN
PRINT AT y,x;d$;AT 0,7+f/14.3;"
": LET d$=" ": LET x=x-1: LET g$
="█": LET f=f-1: GO TO 40
  20 IF INKEY$="7" AND x<30 THEN
 PRINT AT y,x;d$;AT 0,7+f/14.3;"
 ": LET d$=" ": LET g$="█": LET
x=x+1: LET f=f-1: GO TO 40
  25 IF INKEY$="9" AND y>1 THEN
PRINT AT y,x;d$;AT 0,7+f/14.3;"
": LET d$=" ": LET g$="█": LET y
=y-1: LET f=f-1: GO TO 40
  30 IF INKEY$="8" AND y<20 THEN
 PRINT AT y,x;d$;AT 0,7+f/14.3;"
 ": LET d$=" ": LET g$="█": LET
y=y+1: LET f=f-1
  40 IF ATTR (y,x)<7 THEN LET s=
s+1: PRINT AT y,x; INK 6;"█";AT
0,28;s: BEEP .05,3: BEEP .03,5:
BEEP .02,1+(s/4): BEEP .06,1+(s/
3): INK 7: PRINT AT y,x;"█": GO
TO 43
  42 IF SCREEN$ (y,x)="´" THEN L
ET d$="´"
  43 PRINT AT y,x;g$: BEEP .001,
RND*10+15
  47 IF f=0 THEN PRINT AT 0,6;"
";AT 10,11; FLASH 1;" GAME OVER
": BEEP 1,0: BEEP 1,-1: BEEP 1,-
1.8: BEEP 2,-3: GO TO 4
  50 LET g$=CHR$ (149+RND*1): GO
 TO 15+100*(s=r)
 115 FOR y=1 TO 13: BEEP .06,y*2
: PRINT AT 3+y,7; INK 0+y/2; FLA
SH 1;" ¡¡LO CONSIGUIO!! ": NEXT
y
 116 IF t>=39 THEN PRINT AT 19,2
;"Envia clave CHATACODE2017 a:";
AT 20,6; INK 5;"eurocamsuite@yah
oo.es"
 118 PRINT AT 18,2; INVERSE 1;"
 pulse SPACE para iniciar  ": GO
 TO 5500
```

Líneas de la 1 a la 14

Como puedes ver, las líneas del programa están seguidas y, aunque herramientas como ***BASINc*** nos permiten una renumeración automática (incluso actualiza los GOTO's), he decidido dejarlo así buscando una compactación máxima del código y algo de velocidad. En la sección de optimizaciones y pulimentación podrás conocer algunas sencillas técnicas para mejorar el rendimiento de tus juegos escritos en ZXBASIC. En esta primera zona de código aprovechamos para cargar los GDUs (GOSUB 1000), modificamos el juego de caracteres (GOTO 2000), diseño y presentación del menú (GOSUB 900), inicialización de variables, colores generales de papel y tinta (desde que probé mi primer juego *Túneles Marcianos* no concibo un juego que no tenga el PAPER a 0), etc. Todo ello sirviéndonos de los cómodos saltos ofrecidos por las órdenes GOTO (salto sin retorno) y GOSUB (salto a SubRutina con retorno) pese a que algunos pensadores hayan afirmado que tales comandos mutilan de forma irreversible el cerebro de los programadores ;) ... GO TO for ever!!

Líneas de la 15 a la 118

Entre la línea 15 y la 50 encontramos el verdadero núcleo del juego. Es lo que hoy día tal vez denominaríamos *MOTOR* gráfico. Se trata básicamente de un bucle cerrado (por una orden GOTO nuevamente;) que solo podrá finalizar cuando se cumplan unas determinadas condiciones. Para ello, es necesario incluir en dicho bucle los condicionales que comprobarán en todo momento si esas condiciones se cumplen. La línea 50 incluye el salto condicional que salta a la línea 15 (repite bucle motor) ó a la 115 si se cumple la igualdad (s=r), y sin usar la orden IF...THEN para una mayor velocidad. El resto de contenidos que completan el juego (menú, instrucciones, gráficos, carga de GDUs, etc.) quedarán siempre fuera del "motor/bucle" principal para que no ralentice el bucle del motor.

```
 900 GO SUB 1500: INK 7: BEEP .1
,15: PRINT AT 4,0; INVERSE 1;"
■ EL CHATARRERO  GALACTICO ■  ":
 BEEP .1,15: PRINT AT 20,0; INK
1; BRIGHT 0;"FINDEs retro by MaR
aF SOFT©©2017": BEEP .2,20: INK
5: BRIGHT 1: PRINT AT 16,2;"Entr
e nivel de juego (1-4):"; INK 7;
 FLASH 1;" "
 901 PRINT AT 21,21; PAPER 0; IN
K 2;"◢"; PAPER 2; INK 6;"◢"; PAP
ER 6; INK 4;"◢"; PAPER 4; INK 5;
"◢"; PAPER 5; INK 0;"◢";AT 19,23
; PAPER 0; INK 2;"◢"; PAPER 2; I
NK 6;"◢"; PAPER 6; INK 4;"◢"; PA
PER 4; INK 5;"◢"; PAPER 5; INK 0
;"◢"
 902 PRINT AT 7,6; INK 3;"(1) NO
VATO";AT 9,7; INK 4;"(2) INICIAD
O";AT 11,8; INK 5;"(3) EXPERTO";
AT 13,9; INK 6;"(4) SUPERMEGA PR
O*";AT 0,0;"*GUIA RAPIDA DE DESA
RROLLO PARA ZX-SPECTRUM superand
o el nivel 4"
 903 PRINT PAPER 7; BRIGHT 0;AT
18,0;"                          ";
 PAPER 2;" "; PAPER 6;" "; PAPER
 4;" "; PAPER 5;" "; PAPER 7;"
  "
 904 LET p$=INKEY$: LET c=RND*5+
66: POKE 22809,c: POKE 22810,c:
POKE 22811,c: POKE 22841,c: POKE
 22842,c: POKE 22843,c: POKE 228
73,c: POKE 22874,c: POKE 22875,c
: PRINT AT 13,9; INK c-66;"(4) S
UPERMEGA PRO*";AT 0,0;"*"
 905 IF p$<"1" OR p$>"5" THEN GO
 TO 904
 906 LET t=VAL (p$): REM nivel
 907 CLS : PRINT AT 1,0;"Debes
guiar  al antagonista  depacman,
 actualmente en  delicadasituaci
on  de  desempleo,  en elprimer
dia de su nuevo trabajo.";'';"Pu
ede que recoger toda la basuraes
pacial (✻/✹) de cada sector dela
 galaxia no sea tan trepidanteco
mo otras aventuras del pasado,pe
ro de ello depende tu futuro yel
 de toda la RetroGalaxia..";'';"
¡¡SECTOR 4 INCLUYE RECOMPENSA!!!
"
```

```
 908 GO SUB 1300
 909 INK 4: BRIGHT 1: PRINT AT 1
7,1;"6>Izda 7>Dcha 8>Abajo 9>Arr
iba";AT 18,1; INK 3;"(puede usar
 JOYSTICK Sinclair)"; INK 7;AT 2
0,1; INVERSE 1;"PULSE UNA TECLA
 PARA COMENZAR": INVERSE 0: BRIG
HT 1
 910 FOR n=0 TO 20: PRINT AT 15,
4+n;" "; INK 2+n/5;CHR$ (144+n/5
);" ": BEEP .001,RND*10+25
 911 LET p$=INKEY$: IF p$<>"" TH
EN CLS : RETURN
 912 NEXT N
 914 FOR n=0 TO 20: PRINT AT 15,
24-n;" "; INK 2+n/5;CHR$ (144+n/
5);" ": BEEP .001,RND*10+20
 915 LET p$=INKEY$: IF p$<>"" TH
EN CLS : RETURN
 916 NEXT N
 920 GO TO 910
 999 REM return
1000 REM *CODIGO GENERADO POR ZX
-DRAW(cc)2017*
1004 REM GDU IZDA/A/■
1006 POKE USR CHR$ (144),126
1008 POKE USR CHR$ (144)+1,255
1010 POKE USR CHR$ (144)+2,39
1012 POKE USR CHR$ (144)+3,39
1014 POKE USR CHR$ (144)+4,255
1016 POKE USR CHR$ (144)+5,143
1018 POKE USR CHR$ (144)+6,255
1020 POKE USR CHR$ (144)+7,171
1022 REM GDU DCHA/B/■
1024 POKE USR CHR$ (145),126
1026 POKE USR CHR$ (145)+1,255
1028 POKE USR CHR$ (145)+2,228
1030 POKE USR CHR$ (145)+3,228
1032 POKE USR CHR$ (145)+4,255
1034 POKE USR CHR$ (145)+5,241
1036 POKE USR CHR$ (145)+6,255
1038 POKE USR CHR$ (145)+7,213
```

Líneas de la 900 a la 907

Llegamos a la línea **900** desde un salto en la línea **4 GOSUB 900** y justo ahí volvemos a saltar!! con otro GOSUB a la línea **2000**... Algunos programadores de mente fuertemente estructurada estarán pensando ya ¡¡OHH DIOS MIO CREO QUE ESTOY A PUNTO DE VOMITAR!!... Pero ¿Estamos locos? Quizá no tanto. Me explico. En la línea **900** nos disponemos alegremente a preparar el *menú principal* del juego, pero como el dibujo del fantasma en alta resolución requiere bastante código extra y probablemente se nos ocurrió en algún momento de nuestro desarrollo en el que ya no disponíamos de suficientes líneas libres para insertarlo pues decidimos escribir ese código (el que dibuja el fantasma) en algún otro sitio de nuestro programa (línea **2000**) y luego enviar la ejecución allí con un GOSUB **2000**. Aunque algunos puedan tipificarlo de sacrilegio no hay más historia que la que os cuento. Seguimos... La línea **901** dibuja las bandas diagonales de colores con efecto *3D*. La **904** y **905** forman un bucle cerrado (mediante GOTO **904**) del que no se sale hasta pulsar una tecla comprendida entre 1 y 5, cuyo valor determinará el nivel de juego y por tanto el número de residuos (línea **906**) que deberá recoger el *Chatarrero Galáctico*. La **907** muestra instrucciones.

Líneas de la 908 a la 1038

En la línea **908** se vuelve a producir un salto a la **1300** (GOSUB 1300) por similares motivos a los que explicaba en el párrafo anterior. En este caso se trataba de trazar una caja/marco en la pantalla con algunos adornos. Ya finalizado todo el aspecto visual y mientras se muestra la pantalla de instrucciones previa al inicio del juego, entre las líneas **910** y la **920** se establece otro bucle (**920 GOTO 910**) que muestra a nuestro fantasma protagonista moviéndose de un lado a otro y comprueba si pulsemos una tecla para dar comienzo a nuestra apasionante aventura;-)). Observa que las líneas **910** y la **914** son prácticamente idénticas y solo difieren en el valor de la coordenada X que posiciona en pantalla al sprite *GDU* de nuestro protagonista, para que de este modo se produzca el paseo del chatarrero de un lado a otro de la pantalla. Obviamente, el código es mejorable ya que se han creado dos bucles distintos para conseguir la animación del chatarrero hacia ambos lados. Además, cada bucle incluye su propio código que son, el sonido del motor y el condicional que detecta la pulsación de cualquier tecla y que sirve para salir del bucle y continuar con la ejecución del programa. A partir de la **1000** se cargan en memoria los *GDUs* (sprites de 8x8 píxeles).

```
1040 REM GDU ARIBA/C/
1042 POKE USR CHR$ (146),126
1044 POKE USR CHR$ (146)+1,219
1046 POKE USR CHR$ (146)+2,153
1048 POKE USR CHR$ (146)+3,255
1050 POKE USR CHR$ (146)+4,255
1052 POKE USR CHR$ (146)+5,195
1054 POKE USR CHR$ (146)+6,255
1056 POKE USR CHR$ (146)+7,219
1058 REM GDU ABAJO/D/
1060 POKE USR CHR$ (147),126
1062 POKE USR CHR$ (147)+1,255
1064 POKE USR CHR$ (147)+2,255
1066 POKE USR CHR$ (147)+3,219
1068 POKE USR CHR$ (147)+4,153
1070 POKE USR CHR$ (147)+5,255
1072 POKE USR CHR$ (147)+6,255
1074 POKE USR CHR$ (147)+7,219
1076 REM GDU TRAGA/E/
1078 POKE USR CHR$ (148),126
1080 POKE USR CHR$ (148)+1,255
1082 POKE USR CHR$ (148)+2,153
1084 POKE USR CHR$ (148)+3,255
1086 POKE USR CHR$ (148)+4,231
1088 POKE USR CHR$ (148)+5,231
1090 POKE USR CHR$ (148)+6,255
1092 POKE USR CHR$ (148)+7,219
1094 REM GDU MIRA/IZD/F/
1096 POKE USR CHR$ (149),126
1098 POKE USR CHR$ (149)+1,255
1100 POKE USR CHR$ (149)+2,39
1102 POKE USR CHR$ (149)+3,39
1104 POKE USR CHR$ (149)+4,255
1106 POKE USR CHR$ (149)+5,255
1108 POKE USR CHR$ (149)+6,255
1110 POKE USR CHR$ (149)+7,219
1112 REM GDU MIRA/DCH/G/
1114 POKE USR CHR$ (150),126
1116 POKE USR CHR$ (150)+1,255
1118 POKE USR CHR$ (150)+2,228
1120 POKE USR CHR$ (150)+3,228
1122 POKE USR CHR$ (150)+4,255
1124 POKE USR CHR$ (150)+5,255
1126 POKE USR CHR$ (150)+6,255
1128 POKE USR CHR$ (150)+7,219
```

```
1130 REM GDU combustible/
1132 POKE USR "P",126
1134 POKE USR "P"+1,171
1136 POKE USR "P"+2,215
1138 POKE USR "P"+3,171
1140 POKE USR "P"+4,215
1142 POKE USR "P"+5,175
1144 POKE USR "P"+6,255
1146 POKE USR "P"+7,126
1159 REM GDU diagonal logo/H/
1160 POKE USR "H"+0,0
1161 POKE USR "H"+1,1
1162 POKE USR "H"+2,3
1163 POKE USR "H"+3,7
1164 POKE USR "H"+4,15
1165 POKE USR "H"+5,31
1166 POKE USR "H"+6,63
1167 POKE USR "H"+7,127
1170 REM GDU basura/R/
1171 POKE USR CHR$ (161),20
1172 POKE USR CHR$ (161)+1,110
1173 POKE USR CHR$ (161)+2,181
1174 POKE USR CHR$ (161)+3,94
1175 POKE USR CHR$ (161)+4,107
1176 POKE USR CHR$ (161)+5,158
1177 POKE USR CHR$ (161)+6,106
1178 POKE USR CHR$ (161)+7,20
1179 REM GDU ¡ EXCLAMA/S
1180 POKE USR CHR$ (162),0
1181 POKE USR CHR$ (162)+1,8
1182 POKE USR CHR$ (162)+2,0
1183 POKE USR CHR$ (162)+3,8
1184 POKE USR CHR$ (162)+4,8
1185 POKE USR CHR$ (162)+5,8
1186 POKE USR CHR$ (162)+6,8
1187 POKE USR CHR$ (162)+7,8
1190 REM GDU Q basura2/
1191 POKE USR CHR$ (160),52
1192 POKE USR CHR$ (160)+1,127
1193 POKE USR CHR$ (160)+2,58
1194 POKE USR CHR$ (160)+3,191
1195 POKE USR CHR$ (160)+4,250
1196 POKE USR CHR$ (160)+5,111
1197 POKE USR CHR$ (160)+6,253
1198 POKE USR CHR$ (160)+7,76
1199 RETURN
```

Líneas de la 1040 a la 1128

Se continúa con la carga de GDUs *(gráficos definidos por el usuario)*. En el Basic del ZX-Spectrum,el usuario puede generar hasta 21 GDUs y asignarlos a los códigos ASCII a partir del 144 (que se corresponde con la letra A). Existen métodos para superar este límite, pero su explicación escapa a la finalidad de la presente guía.
Verás que antes de comenzar a definir el gráfico (mediante la órden POKE que escribe en memoria), he incluido una línea de comentario en la que se indica el código ASCII/carácter necesario para acceder a dicho GDU, y además se muestra el GDU para que visualmente nos resulte más cómoda y rápida su interpretación.
La definición de sprites se consigue mediante código binario en el que el valor CERO supone un vacío o píxel apagado, y el UNO lo contrario. En el juego del Chatarrero verás que los sprites representan las distintas posturas o gestos que muestra el protagonista, además de otros gráficos que representan a la basura, la gasolina, o los bloques diagonales usados en el menú principal del juego.

Líneas de la 1130 a la 1199

Aquí se continúa básicamente con lo mismo, y como verás, los GDUs están definidos de diferentes formas, en unos casos se designa el carácter directamente entrecomillado, y en otros mediante el valor numérico ASCII equivalente. El resultado de uno u otro modo es idéntico.
Para acceder a todos estos GDUs debemos activar el modo G desde el teclado del *ZX-Spectrum* (CAPS+9) y luego pulsar la tecla correspondiente al GDU deseado (A, B, C, etc...). Así se accede estos gráficos.
Es típico definir todos los GDUs de nuestro programa al final del código del mismo, y luego saltar a la línea donde comienza la carga de los GDUs mediante un GOSUB, en cuyo caso, siempre deberemos volver mediante la correspondiente orden RETURN.
Como verás, al final de todos los gráficos, en la línea **1199**, una orden *RETURN* devuelve la ejecución del programa justamente a la línea siguiente al GOSUB que nos trajo hasta aquí. Si quieres comprobarlo, verás que en la línea **2** del programa se ordena un salto GOSUB **1000** que es precisamente donde comienza la carga de los GDUs.

```
1300 PLOT 0,0: DRAW 255,0: DRAW
0,175: DRAW -255,0: DRAW 0,-175:
 RETURN
1305 PLOT 7,7: DRAW 241,0: DRAW
0,161: DRAW -241,0: DRAW 0,-161:
 RETURN
1400 REM *** RUTINA MARCO TOTAL
1401 LET IC=185
1402 INK 7: PLOT 0,0: DRAW BRIGH
T 0;255,0: DRAW BRIGHT 0;0,175:
DRAW BRIGHT 1;-255,0: DRAW BRIGH
T 1;0,-175: PLOT 1,1: DRAW BRIGH
T 0;253,0: DRAW BRIGHT 0;0,173:
DRAW BRIGHT 1;-253,0: DRAW BRIGH
T 1;0,-173: PLOT 3,3: DRAW BRIGH
T 0;249,0: DRAW BRIGHT 0;0,169:
DRAW BRIGHT 1;-249,0: DRAW BRIGH
T 1;0,-169
1403 PLOT INK 2; BRIGHT 1;IC,0:
PLOT INK 2; BRIGHT 0;IC+8,0: PLO
T INK 2; BRIGHT 1;255,64: PLOT I
NK 2; BRIGHT 0;255,56
1404 PLOT INK 6; BRIGHT 1;IC+16,
0: PLOT INK 6; BRIGHT 0;IC+24,0:
 PLOT INK 6; BRIGHT 0;255,40: PL
OT INK 6; BRIGHT 1;255,48
1405 PLOT INK 4; BRIGHT 1;IC+32,
0: PLOT INK 4; BRIGHT 0;IC+40,0:
 PLOT INK 4; BRIGHT 0;255,24: PL
OT INK 4; BRIGHT 1;255,32
1406 PLOT INK 5; BRIGHT 1;IC+48,
0: PLOT INK 5; BRIGHT 0;IC+56,0:
 PLOT INK 5; BRIGHT 1;255,16
1409 RETURN
1500 REM DIBUJA FANTASMA
1501 INK 3: BRIGHT 1
1503 PLOT 200,90: DRAW 0,15: PLO
T 200,105: DRAW 2,2: PLOT 219,90
: DRAW 0,15: PLOT 219,105: DRAW
-2,2: PLOT 217,107: DRAW -15,0:
PLOT 200,90: DRAW 19,0
1505 CIRCLE 206,100,2: CIRCLE 20
6,100,1: CIRCLE 213,100,1: CIRCL
E 213,100,2
1515 PLOT 207,94: DRAW 5,0
1517 PLOT 215,91
1521 PLOT 204,91
1525 PLOT 210,91
1529 PLOT 215,92
1533 PLOT 204,92
1599 RETURN
```

```
2000 CLEAR 54999
2003 LET mc=59000: LET chr=55000
2004 FOR f=mc TO mc+26: READ a:
POKE f,a: NEXT f
2006 DATA 17,0,200,237,83,54,92,
33,0,60,126,18,254,60,32,+3,62,1
26,18,19,35,124,254,64,32,-16,20
1
2007 RANDOMIZE chr: POKE mc+1,PE
EK 23670: POKE mc+2,PEEK 23671:
LET l=USR mc
2010 GO TO 4
5500 REM supense melody
5503 LET t=.2
5504 FOR n=1 TO 4
5506 BEEP t,0: GO TO 4+5504*(INK
EY$<>" ")
5508 BEEP t,5: GO TO 4+5506*(INK
EY$<>" ")
5510 BEEP t,8: GO TO 4+5507*(INK
EY$<>" ")
5511 NEXT n
5514 FOR n=1 TO 4
5516 BEEP t,1: GO TO 4+5514*(INK
EY$<>" ")
5518 BEEP t,5: GO TO 4+5516*(INK
EY$<>" ")
5520 BEEP t,8: GO TO 4+5517*(INK
EY$<>" ")
5521 NEXT n
5524 FOR n=1 TO 4
5526 BEEP t,-2: GO TO 4+5524*(IN
KEY$<>" ")
5528 BEEP t,5: GO TO 4+5526*(INK
EY$<>" ")
5530 BEEP t,7: GO TO 4+5527*(INK
EY$<>" ")
5531 NEXT n
5590 GO TO 5504
```

Líneas de la 1300 a la 1599

La línea **1300** es en realidad una SUBRUTINA en la que se incluye el código para dibujar un marco en alta resolución y como ves, también integra ya la orden RETURN al final de la línea, para que así podamos llamar a esta SUBRUTINA desde cualquier parte de nuestro programa cada vez que neceitemos dibujar un marco (GOSUB 1300)
La línea **1305** es similar a la anterior en su funcionalidad, solo que dibuja un marco algo más pequeño y consigue el efecto de marquesina de doble línea.
La línea **1400** inicia la rutina encargada de trazar la marquesina doble con filigranas de color (arcoiris spectrum) en la esquina inferior derecha. Esta marquesina es la utilizada en el escenario principal del juego.
Las líneas **1500-1599** están destinadas a trazar la imagen en alta resolución del chatarrero galáctico (fantasma) que se muestra en el menú principal del juego.

Líneas de la 2000 a la 5590

En la línea **2000** hemos implementado una pequeñísima rutina en código máquina que modifica el juego de caracteres por defecto de nuestro ZX-Spectrum. No quiero decir con esto que la fuente del ordenador milagroso sea fea ni mucho menos, simplemente es un experimento más que muestra las enormes posibilidades de pequeñas rutinas de código máquina y que podemos integrar fácilmente en nuestros programas escritos en BASIC sin ningún tipo de conocimiento de ensamblador y mejorando su atractivo. A partir de la línea **5500** se inicia el bucle final, con música incluida, que se muestra en pantalla al superar cualquier nivel del juego. Los GOTO's con salto condicional permiten seguir la secuencia de ejecución de la música y comprobar en cada línea si se pulsa la tecla ESPACIO, en cuyo caso el valor numérico al que saltará el GOTO es **4**, es decir, el inicio del juego nuevamente pero sin volver a cargar el juego personalizado de caracteres ni los GDUs. Esta técnica, algo enrevesada visualmente, nos permite ejecutar la música y detectar la pulsación de la tecla para romper el bucle, teniendo en cuenta que el comando BEEP del ZX-Spectrum congela literalmente el procesador Z80 mientras suena.

09
Mapa Estructural del Código

Tras la sección anterior, que igual te ha saturado un poco si te la estudias de sopetón, vamos a tratar algo más liviano huyendo de un posible colapso cognitivo.

A menos que seas extremadamente meticuloso y purista en la fase de análisis de tu juego o quieras dedicarle tiempo al estudio de la creación previa de aburridos organigramas (también llamados diagramas de flujo) antes de entrar en la fase de picado de código, es casi seguro que en algún momento debas enfrentarte a un soberbio problema de organización de tu código, por eso esta sección te será de gran ayuda para poder superarlo sin un sólo diagrama de flujo. Me explicaré mejor.



El día que escribí el borrador de este juego en mi *Spectrum* +3 el código inicial resultó en un tamaño aproximado de 1 *KiloByte* (estoy convencido de que podía correr perfectamente en un *ZX-81* con mínimas modificaciones) pero finalizado el proceso

de mejoras gráficas y funcionalidades para convertir nuestro juego en algo medianamente "serio", en el momento de desarrollar la presente guía rápida *(basada en la versión 0.37b)* el código ha engordado hasta aproximadamente unos 11 *KBytes* por lo que, dicho sea de paso y al menos en teoría, aún deberíamos poder compilarlo y ejecutarlo sin problemas en un *ZX-Spectrum 16K*.

Sin embargo, y en contra de lo que muchos puedan imaginar, estos aparentemente ridículos 11 KB de código suponen ya un tamaño considerable de texto para el limitado editor de nuestro *ZX-Spectrum* y, para más inri, el juego se ha desarrollado desde el principio sin ninguna previsión sólida y lo que algunos programadores denominan *código espagueti* se convierte aquí en una característica de facto. Debes saber que esto te ocurrirá a menudo a no ser que seas muy escrupuloso con la elaboración de tu código, pero de todos modos, no está de más recordar siempre que herramientas como *BASINc* te permitirán reestructurar el código de tu programa cómodamente renumerando las líneas de forma automática, y aunque ello tampoco significa que quede perfectamente estructurado, al menos podremos introducir cambios y añadir comentarios si lo necesitáramos.

En este sentido y en relación con el citado *código espagueti*, debes saber que han sido ríos de tinta los que se han

vertido para denostar al antiguo *BASIC* lineal por esta peculiaridad que otros lenguajes de su época (*Pascal, C, Cobol...)* no ofrecían y que permitía al programador *BASIC*, novato y/o autodidacta donde los haya, saltar de un lado a otro como le venía en gana sin ningún tipo de restricciones, y esa madeja de *GOTOs* y saltos sin control aparente acabó convirtiéndose en un agravio que acompañó al *BASIC* desde prácticamente sus orígenes remotos, salpicando injustificadamente a todas las variedades posteriores de este hermoso y amigable lenguaje. Incluso cuando *BASIC* evolucionó y se transformó en potentes lenguajes más estructurados (*AmigaBASIC, SuperBasic* de *Sinclair-QL*) e incluso compilados (*QuickBasic, TurboBASIC*, etc) enterrando para siempre el sistema lineal de ejecución, la calumnia siguió pesando como una losa para este lenguaje de programación al que la informática en general debería estar eternamente agradecida.

Sin embargo, a pesar de todo este tostón que te estoy dando con la historia del *código BASIC espagueti*, veremos como un sencillo esquema de la estructura de nuestro programa puede ayudarnos a resolver la madeja y mantener al malévolo *código espagueti* bajo control.

Para el caso de nuestro juego tipo *EL CHATARRERO GALÁCTICO* el esquema de la siguiente página podría servirnos perfectamente:

Línea	Descripción general del código
1-14	Inicialización general con llamadas a subrutinas ➡ *incluye salto a **1000** para cargar sprites GDUs.* ➡ *incluye salto a **2000** para cambiar tipo de letras.* ➡ *incluye salto a **900** para presentar menú de opciones.* ➡ *incluye salto a **1400** para dibujar escenario de juego (marquesina doble con espectro de colores).*
15-50	Núcleo del juego en bucle cerrado (motor) con GOTO condicional
115-118	Comprueba si gana y corresponde premio ➡ *incluye salto a 5500 para mantener melodía final.*
900	*[SUBRUTINA]* Muestra Menú de inicio e instrucciones en bucle cerrado. De la línea **907** a **920** se muestra instrucciones de juego en bucle cerrado y al pulsa tecla se retorna al programa principal para iniciar juego. ➡ *incluye al inicio salto a **1500** para dibujar fantasma en alta resolución*
1000-1199	*[SUBRUTINA]* Carga sprites GDU's en memoria
1300-1305	*[SUBRUTINAS]* Se trazan marcos en alta resolución para pantalla de instrucciones (simple) y para pantalla de juego (doble).
1400	*[SUBRUTINA]* Trazan marco de doble línea con filigranas de colores en esquina inferior derecha y que se utiliza en el escenario principal del juego
1500	*[SUBRUTINA]* Dibuja fantasma del menú en alta resolución
2000	*[SUBRUTINA] Rutina* en código máquina que modifica juego caracteres del *Spectrum*
5500	*[SUBRUTINA]* Melodía final de tipo cíclica que suena al superar nivel y que detecta pulsación de tecla para salir del bucle y reiniciar juego cargando el menú principal

Sinceramente pienso que este breve "*esquema estructural*" puede sustituir de algún modo a los odiados y aburridos diagramas de flujo que otrora quitaron el sueño a algunos estudiantes de informática. Como verás, puedes hacerlo de cualquier manera con la única condición de que te sirva realmente a la hora de corregir, modificar o ampliar el código.

Ya para terminar te diré que esta práctica no es ni mucho menos exclusiva del desarrollo de videojuegos y puedes hacerla extensible a cualquier tipo de desarrollo cuyo tamaño o complejidad lo requiera. Y es que mantener actualizado un sencillo esquema de la estructura de cualquier programa siempre nos ayudará a mantener bajo control el *código espagueti* y, sobre todo, a retomar el desarrollo de nuestros programas (después de meses o incluso años) con realtiva normalidad y sin demasiados traumas. Es posible que en los modenos lenguajes estructurados este problema se haya superado ampliamente mediante el uso de funciones modulares o subrutinas que son llamadas por su nombre y también por la restricción del comando/orden *GOTO* a un uso prácticamente residual, pero en ***ZXBASIC*** resulta indispensable... *GO TO* for ever!!

10
Análisis Profundo del Código

Y ahora vamos a entrar ya en materia directamente con una vieja fórmula didáctica que consiste en ir analizando, una a una, todas las líneas del código de nuestro programa detallando el efecto que surte. La estructura del formato que seguiremos no puede ser más simple y si al principio puede parecer que te pierdas con algunos saltos de línea, si echas un ojo al esquema de la sección anterior (***ESTRUCTURA DEL CÓDIGO)*** te reecontrarás rápidamente.

Dicho esto, creo que ya estamos preparados para despegar, así que... Bon voyage mon ami!!

```
   1 REM **** EL CHATARRERO GALA
CTICO ****
```

- **Línea 1:** Casi tan popular como el **HOLA MUNDO!**, esta es una de las primeras líneas que debía aprender cualquier jovencito usuario de *Spectrum*. El comando **REM** sirve para incluir cualquier comentario en el código de nuestro programa, y el nombre del juego parece ser buena idea. Todo lo que se escribe tras dicho comando es ignorado por el programa y únicamente nos sirve a modo de apunte o aclaración.

La línea **1** es de tipo simple pues solo incluye un comando u orden, pero como verás, el ***ZXBASIC*** permite el uso de líneas múltiples (con varias órdenes o comandos) separándolas mediante dos puntos **:**. Esto es útil pues ahorra algo de memoria y puede mejorar la velocidad de ejecución.

```
   2 PAPER 0: BORDER 0: BRIGHT 0
: CLS : RANDOMIZE : GO SUB 1000:
 REM carga SPRITEs: A= B= C=
D= E= F= G= H= P= Q= R=
S=¡
```

Línea 2: (de tipo múltiple) establecemos el color del fondo y el borde de la pantalla a cero, que equivale al negro, así como el atributo de brillo por defecto que, al establecer a cero queda desactivado (el brillo solo puede tener dos valores, **0** ó **1**). La orden/comando **RANDOMIZE** sirve para reinicializar la semilla aleatoria y evitar así que se repitan las sucesiones de números aleatorios por ejemplo, al situar objetos en pantalla. A continuación se encuentra una orden de salto que envía la ejecución del programa (o *punto de ejecución*[14]) a una subrutina situada en la línea **1000** mediante el comando/orden **GOSUB 1000** y que finaliza en la línea **1199** con la orden **RETURN**. Tras ejecutarse la subrutina, con la orden **RETURN** el control del programa o punto de ejecución vuelve a la orden o

14 **PUNTO DE EJECUCIÓN:** *es clave en programación y para lo sucesivo comprender el significado de esta expresión referida al lugar concreto del programa que se está ejecutando en cada momento, y aunque los tiempos de ejecución puedan ser rapidísimos, no olvidemos que el procesador de nuestra computadora solo puede ejecutar un proceso o comando en cada momento. De esta forma, podemos decir por tanto que las órdenes de salto alteran el orden lógico (secuencial) del punto de ejecución de un programa.*

línea inmediatamente posterior a la orden **GOSUB** que ordenó el salto. A continuación se introduce una orden **REM** para indicarnos que el salto a la rutina se hará para cargar los gráficos definidos por el usuario (GDUs) o sprites (en su momento ya veremos como funcionan). También aprovechamos este **REM** para mostrar los GDUs que se cargan exactamente y en que tecla gráfica quedan asignados cada uno de ellos. Esto es muy útil pues nos sirve a modo de índice.

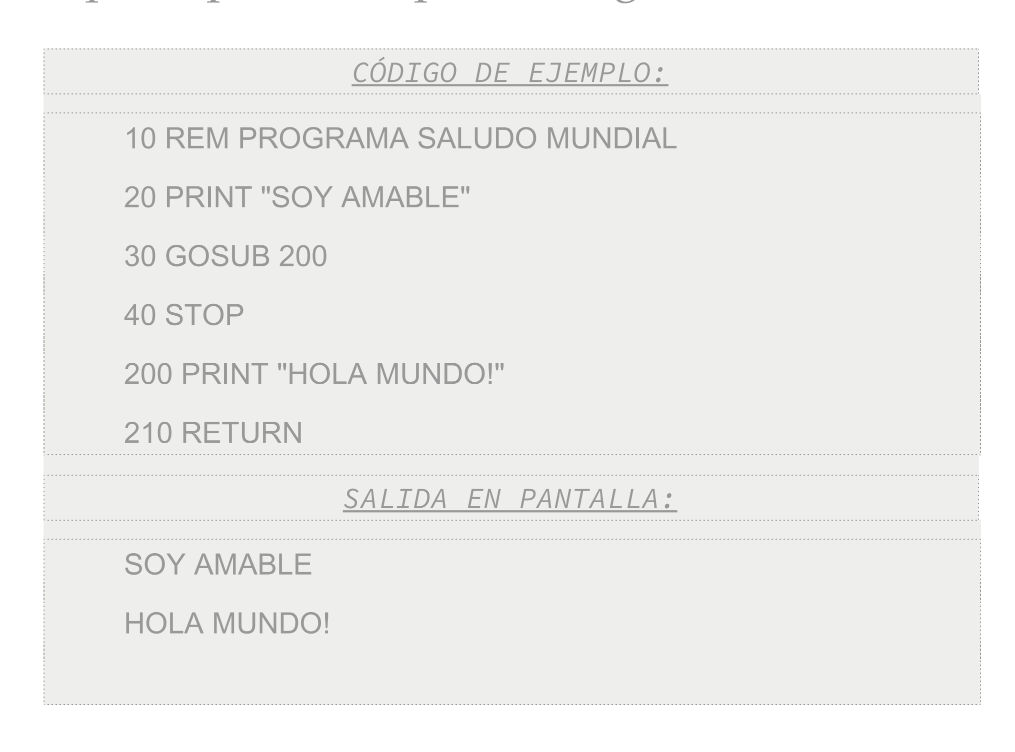
```
   3 GO TO 2000: REM FONTs- solo
 en modo 48K
```

Línea 3: no debe ejecutarse mientras trabajemos en modo 128 ya que sirve para cargar una pequeña rutina de código máquina que personaliza el tipo de letra de nuestro Spectrum y da problemas en modo 128. Lo mejor mientras trabajas con el programa es ignorarla anteponiendo simplemente una orden **REM** antes del **GOTO**. La orden **REM** nos sirve a modo de advertencia y recordarnos este problema.

```
   4 CLS : GO SUB 900: GO SUB 13
05: GO SUB 1400: REM PARA REINIC
IAR JUEGO GOTO 4
```

Línea 4: **CLS** nos sirve para borrar la pantalla. Las órdenes **GOSUB** sirven para saltar al número de línea indicado y regresar tras ejecutar una subrutina. Todas las subrutinas se finalizan con la orden **RETURN**, que devuelve el control del programa (punto de ejecución) a la orden inmediantemente

posterior al **GOSUB** que llamó a la rutina. Una rutina (también llamada subrutina) es un trozo de código que queremos separar del resto del programa (por legibilidad y estructuración) para llamarlo cuando nos interese. Vamos a ver un ejemplo muy sencillo para que no te quede ninguna duda:



CÓDIGO DE EJEMPLO:

```
10 REM PROGRAMA SALUDO MUNDIAL
20 PRINT "SOY AMABLE"
30 GOSUB 200
40 STOP
200 PRINT "HOLA MUNDO!"
210 RETURN
```

SALIDA EN PANTALLA:

```
SOY AMABLE
HOLA MUNDO!
```

- *En este ejemplo, la orden* **RETURN** *de la línea* **210** *devuelve el control del programa o punto ejecución a la línea inmediatamente posterior al* **GOSUB 200**. *Si se tratase de una línea múltiple (con varias acciones/órdenes separadas por :) la orden* **RETURN** *devolvería el control o punto de ejecución a la siguiente orden dentro de la misma línea que se estaba ejecutando y desde la que se ordenó la orden* **GOSUB**. *Las rutinas en* ***ZXBASIC*** *funcionan así, con una llamada o salto hacia la rutina, y, una vez finalizada la ejecución de la rutina y encontrada la orden* **RETURN** *, otro salto de retorno para seguir la ejecución*

del programa desde donde se quedó.

Aclarado esto sigamos con lo nuestro. El primer salto **GOSUB 900** *nos lleva al bloque de código encargado de mostrar el menú principal del juego y esperar hasta que se pulse una tecla seleccionando el nivel de juego, tras pulsar/selccionar el nivel de juego, se muestra la pantalla de instrucciones y se espera a que se pulse otra tecla para continuar la ejecución del programa. Cuando se finaliza toda la rutina y se encuentra una orden* **RETURN** *(línea* **915***), el punto de ejecución vuelve a la línea* **4** *y continúa con la segunda orden que es* **GOSUB 1305,** *encargada de llamar a la subrutina que dibuja un marco doble (de dos líneas) en alta resolución en la pantalla de juego. Finalizada esta rutina se retorna de nuevo a la línea* **4** *para volver a saltar con* **GOSUB 1400***, rutina que dibuja y completa toda la marquesita que verás durante el juego y que presenta un embellecedor efecto arcoiris en la zona inferir derecha. Esta rutina finaliza en la línea* **1409** *con la orden* **RETURN.** A continuación una orden **REM** nos recuerda que, una vez compilado el código mediante el compilador MCODER3, si queremos reiniciar el juego debemos hacerlo mediante **GOTO 4** y no con la orden ***RUN***.

```
   5 PRINT #1; INK T+2; BRIGHT 0
; INVERSE 1;"EL CHATARRERO GALAC
TICO ▌"; BRIGHT 1;"SECTOR:";t
```

Línea 5: Activa la impresión por pantalla en el canal 1. Esto es para poder aprovechar las dos líneas inferiores de la pantalla de nuestro Spectrum, exactamente las que usamos para introducir el código cuando trabajamos en modo 48K. Estas dos líneas inferiores no pueden utilizarse como el resto de la pantalla mediante coordenadas, por ello es necesario usar el modificador **#1** en la orden **PRINT**. Verás que toda la línea consta de una sola orden **PRINT** seguida de varios modificadores o parámetros separados por punto y coma. Vamos a verlo. La orden **INK** puede usarse de forma independiente o como operador anexo a una orden **PRINT** ó **PLOT**, y su función es indicar al ordenador qué color (de la paleta de 8 colores) debe usar para mostrar en pantalla lo que el comando **PRINT** ó **PLOT** ordenen. En nuestro caso, la orden **INK** va seguida de una variable llamada **T** y que viene ya determinada por el nivel de juego seleccionado previamente en el menú principal. La variable **T** representa el sector y puede adoptar valores entre 1 y 4. Como verás, a ese valor contenido en la variable **T** le hemos sumado 2 unidades para conseguir un rango de colores más adecuado (más visibles), de forma que del rango de 1 a 4 hemos pasado a otro de 3 a 6. Este pequeño detalle nos sirve para diferenciar las colores que se mostrarán en pantalla en función

del nivel de juego y usados con ingenio pueden contribuir a mejorar la experiencia del jugador. Las órdenes restantes **BRIGHT 0** e **INVERSE 1** complementa también la orden **PRINT** desactivando el brillo (solo puede ser 0 ó 1) y activando la función inversa. Ambas admiten solo dos estados. La función **INVERSE** intercambia los colores establecidos para el fondo del texto y el texto en sí. Mira este ejemplo: si activo un color negro para el texto mediante la correspondiente orden **INK 0** y un color de fondo blanco mediante la orden **PAPER 7**, a continuación invocamos a la función **INVERSE 1**, el resultado será éste. Tras cada punto y coma, vamos pasando valores a la orden **PRINT**, los valores entre comillados se imprimen tal cual, literalmente, y las variables, que no van entrecomilladas, se sustituyen por el valor que posean en ese momento. Ah! otra cosa, simpre que usamos punto y coma después de imprimir algo, la siguiente impresión se hace justo a continuación, sin salto de línea. Si no indicamos punto y coma al final, en la próxima orden **PRINT** se comienza con un salto de línea o retorno de carro.

```
   6 INK 7: BRIGHT 1: FOR n=1 TO
 45: PRINT AT INT (RND*19)+1,INT
 (RND*29)+1;"´": NEXT n
```

Línea 6: Establece el color blanco para tinta (se refiere al texto). La orden **PAPER** se refiere al fondo o papel. Se activa el brillo. Se inicia un bucle. Bievenidos al maravilloso mundo de la iteración. Los

bucles en ***ZXBASIC*** son sencillamente geniales y muy sencillos, en este caso debes tener en cuenta dos cosas muy básicas: un bucle se usa para repetir algo varias veces, y todo bucle está formado por dos partes, la primera es el **FOR....TO** en la que se establecen las características del bucle, y la segunda la orden **NEXT** que cierra el bucle. Todo el código que hay entre ambos puntos es lo que se repetirá el número de veces que el bucle indique.

```
   7 LET r=0: LET cb=t*5: BRIGHT 0: REM contador de basura; numero de basuras=nivel t*10
```

```
   8 FOR n=1 TO cb: LET x=RND*29+1: LET y=RND*19+1: IF ATTR (y,x)>6 THEN PRINT AT y,x; INK INT (n/5)+1;"❋": LET r=r+1
```

```
   9 NEXT n
```

```
  10 FOR n=1 TO cb: LET x=RND*29+1: LET y=RND*19+1: IF ATTR (y,x)>6 THEN PRINT AT y,x; INK INT (n/5)+1;"❋": LET r=r+1
```

```
  11 NEXT n
```

```
  12 PRINT AT 0,0; INK 7; BRIGHT 1; INVERSE 1;" FUEL:"; INVERSE 0;"    "; INVERSE 1;"
 SCORE:"; INVERSE 0;"0   "; INVERSE 1;" "
```

```
  13 LET t=t*10: LET s=0: LET f=
150: LET x=15: LET y=12: LET g$=
"█": LET d$=" ": PRINT BRIGHT 1;
AT 0,6; INK 2;"▒▒"; INK 6;"▒▒▒";
 INK 4;"▒▒▒▒▒▒ "
```

```
  14 INK 7: BRIGHT 1: REM color
nave cambia a 6 al comer
```

```
  15 IF INKEY$="6" AND x>1 THEN
PRINT AT y,x;d$;AT 0,7+f/14.3;"
": LET d$=" ": LET x=x-1: LET g$
="█": LET f=f-1: GO TO 40
```

```
  20 IF INKEY$="7" AND x<30 THEN
 PRINT AT y,x;d$;AT 0,7+f/14.3;"
 ": LET d$=" ": LET g$="█": LET
x=x+1: LET f=f-1: GO TO 40
```

```
  25 IF INKEY$="9" AND y>1 THEN
PRINT AT y,x;d$;AT 0,7+f/14.3;"
": LET d$=" ": LET g$="█": LET y
=y-1: LET f=f-1: GO TO 40
```

```
  30 IF INKEY$="8" AND y<20 THEN
 PRINT AT y,x;d$;AT 0,7+f/14.3;"
 ": LET d$=" ": LET g$="█": LET
y=y+1: LET f=f-1
```

```
  40 IF ATTR (y,x)<7 THEN LET s=
s+1: PRINT AT y,x; INK 6;"█";AT
0,28;s: BEEP .05,3: BEEP .03,5:
BEEP .02,1+(s/4): BEEP .06,1+(s/
3): INK 7: PRINT AT y,x;"█": GO
TO 43
```

```
  42 IF SCREEN$ (y,x)="'" THEN L
ET d$="'"
```

```
  43 PRINT AT y,x;g$: BEEP .001,
RND*10+15
```

```
  47 IF f=0 THEN PRINT AT 0,6;"
";AT 10,11; FLASH 1;" GAME OVER
": BEEP 1,0: BEEP 1,-1: BEEP 1,-
1.8: BEEP 2,-3: GO TO 4
```

```
  50 LET g$=CHR$ (149+RND*1): GO
 TO 15+100*(s=r)
```

```
 115 FOR y=1 TO 13: BEEP .06,y*2
: PRINT AT 3+y,7; INK 0+y/2; FLA
SH 1;" ¡¡LO CONSIGUIO!! ": NEXT
y
```

```
 116 IF t>=39 THEN PRINT AT 19,2
;"Envia clave CHATACODE2017 a:";
AT 20,6; INK 5;"eurocamsuite@yah
oo.es"
```

```
 118 PRINT AT 18,2; INVERSE 1;"
 pulse SPACE para iniciar  ": GO
 TO 5500
```

```
 120 REM GO TO 4
```

```
 900 GO SUB 1500: INK 7: BEEP .1
,15: PRINT AT 4,0; INVERSE 1;"
  EL CHATARRERO  GALACTICO    ":
 BEEP .1,15: PRINT AT 20,0; INK
1; BRIGHT 0;"FINDEs retro by MaR
aF SOFT©©2017": BEEP .2,20: INK
5: BRIGHT 1: PRINT AT 16,2;"Entr
e nivel de juego (1-4):"; INK 7;
 FLASH 1;" "
```

```
 901 PRINT AT 21,21; PAPER 0; INK 2;"◢"; PAPER 2; INK 6;"◢"; PAPER 6; INK 4;"◢"; PAPER 4; INK 5;"◢"; PAPER 5; INK 0;"◢";AT 19,23; PAPER 0; INK 2;"◢"; PAPER 2; INK 6;"◢"; PAPER 6; INK 4;"◢"; PAPER 4; INK 5;"◢"; PAPER 5; INK 0;"◢"
```

```
 902 PRINT AT 7,6; INK 3;"(1) NOVATO";AT 9,7; INK 4;"(2) INICIADO";AT 11,8; INK 5;"(3) EXPERTO";AT 13,9; INK 6;"(4) SUPERMEGA PRO*";AT 0,0;"*GUIA RAPIDA DE DESARROLLO PARA ZX-SPECTRUM superando el nivel 4"
```

```
 903 PRINT PAPER 7; BRIGHT 0;AT 18,0;"                        "; PAPER 2;" "; PAPER 6;" "; PAPER 4;" "; PAPER 5;" "; PAPER 7;"    "
```

```
 904 LET p$=INKEY$: LET c=RND*5+66: POKE 22809,c: POKE 22810,c: POKE 22811,c: POKE 22841,c: POKE 22842,c: POKE 22843,c: POKE 22873,c: POKE 22874,c: POKE 22875,c: PRINT AT 13,9; INK c-66;"(4) SUPERMEGA PRO*";AT 0,0;"*"
```

```
 905 IF p$<"1" OR p$>"5" THEN GO TO 904
```

```
 906 LET t=VAL (p$): REM nivel
```

```
 907 CLS : PRINT AT 1,0;"Debes
guiar  al antagonista  depacman,
 actualmente en  delicadasituaci
on  de  desempleo,  en elprimer
dia de su nuevo trabajo.";'';"Pu
ede que recoger toda la basuraes
pacial (❋/❋) de cada sector dela
 galaxia no sea tan trepidanteco
mo otras aventuras del pasado,pe
ro de ello depende tu futuro yel
 de toda la RetroGalaxia..";'';"
¡¡SECTOR 4 INCLUYE RECOMPENSA!!!
"
```

```
 908 GO SUB 1300
```

11
Técnicas Básicas de Optimización

En resumen, podemos afirmar que es ésta, básicamente, una historia de pruebas *Benchmarks* para nuestro *Spectrum*.

En esta sección vamos a estudiar como podemos mejorar la velocidad de nuestro programa ***ZXBASIC*** simplemente puliendo algunos detalles o, como diría nuestro buen amigo Juanfra (*EMS*), abrillantando el código con el paño mágico. Atento porque, aunque esta información no venga en los manuales del usuario ni en los libros de programación, los resultados te van a sorprender.

Para nuestro caso, vamos a tomar como código de partida una pequeña rutina empleada en nuestro juego tipo ***EL CHATARRERO GALÁCTICO*** y cuya única finalidad consiste en mostrar en la pantalla de juego los cuerpos celestes o estrellas, simulando así el firmamento en el que deberá desenvolverse nuestro protagonista, de manera que, crono en mano, vamos a intentar mejorar la velocidad de esta sencilla rutina. Como verás, he utilizado diferentes estilos de letra (negrita) de forma alternativa en las

líneas del programa para intentar mejorar un poco la legibilidad del código. Comenzamos:

* OPTIMIZACION DE LA RUTINA CON BUCLE UTILIZADA PARA DIBUJAR CUERPOS CELESTES EN EL FIRMAMENTO.

```
5 PAPER 0: BORDER 0: CLS
10 FOR n=1 TO 200
11 LET x=INT (RND*29)+1
12 LET y=INT (RND*19)+1
15 PRINT AT y,x; INK (RND*4)+3;"'"
18 NEXT n
20 PRINT "the end!"
25 BEEP 1,20
```

Este trozo de código, prácticamente idéntico al recurrido en nuestro juego tipo, imprime 200 caracteres semigráficos ' (comilla simple) a lo largo de todo el ancho y alto de la pantalla, exactamente desde la ***coordenada X*** de valor 1 hasta el 29 (coordenada relativa la situación horizontal) y para la ***coordenada Y*** desde el valor 1 hasta el 19 (coordenada relativa a la altura o situación vertical). Además, establece un rango aleatorio (mediante la función ***RND***) para el color de la tinta con el que impirmirá el símbolo ' (cuerpo celeste del firmamento) que va de 4 a 7 para utilizar solo los colores más brillantes de la paleta. Al finalizar el proceso repetitivo imprime un mensaje en pantalla (*the end!*) y emite un sonido únicamente para ardvertirme que el programa ha concluido y

detener el cronómetro. Obviamente, hay márgenes de error en las siguientes mediciones, pero son relativamente pequeños y no alteran en absoluto el sentido ni la finalidad del experimento. De manera que ...

15,5 sg invierte en la ejecución completa de esta sencilla rutina nuestro querido *Sepectrum*, décima arriba décima abajo.

Toda nuestra optimización se va a centrar ahora en el bucle, que obviamente es el código que más veces se repite en la rutina. Así, entramos ya en materia reagrupando todo el código que encierra el bucle (o sea, entre el comando ***FOR*** y el ***NEXT***) en una única línea múltiple en la que separamos las distintas órdenes por : (dos puntos). Aunque no quede tan claro a la vista suele ahorrar memoria y también algunos ciclos de procesador. La reagrupación de varias órdenes en líneas múltiples suele ser un buen hábito siempre que se utilice en su justa medida.

```
5 PAPER 0: BORDER 0: CLS
10 FOR n=1 TO 200: LET x=INT(RND*29)+1: LET y=INT(RND*19)+1: PRINT
   AT y,x; INK RND*4+3;"": NEXT n
19 BEEP 1,20
20 PRINT "the end!"
```

14,8 sg Bueno, fíjate que con esta leve modificación el mismo proceso ya tarda algo menos. Vale, pensarás que esta primera pasada de

plumero no es para lanzar cohetes pero menos da una piedra. Ahora probemos a prescindir del uso de las variables *X* e *Y* (que hemos empleado como valores de coordenadas para la pantalla) y usamos los valores directos, de este modo vamos a ahorrarle al procesador dos asignaciones (LET) en cada ciclo del bucle. Ya de paso, suprimimos también la función INT que calcula la parte entera de un número pues no resulta imprescindible para el funcionamiento correcto de la orden ***PRINT AT***. Veamos...

```
5 PAPER 0: BORDER 0: CLS
10 FOR n=1 TO 200: PRINT AT RND*19+1,RND*29+1;INK RND*4+3;"": NEXT
   n
19 BEEP 1,20
20 PRINT "the end!"
```

14,2 sg Perfecto. Como estamos usando valores aleatorios con la orden/comando ***INK RND*** para conseguir un spectro/rango de colores determinado (del 4 al 7), ahora vamos a tratar de conseguir idéntico efecto a partir de la variable ***N*** que no es otra cosa que el valor del bucle que va incrementando su valor desde 1 a 200. Todo ello con el objeto de omitir la función ***RND***. Para conseguir el rango de valores que necesitamos para los colores, podemos dividir el valor de N por un número adecuado para que nos de resultados entre 0 y 4, a cuyo resultado le sumaremos 3 para hacerlo

coincidir con el rango deseado de colores (4 a 7) y que el color inicial nunca sea cero, uno, o dos. Ese valor que necesitamos para dividir el bucle será 50, así, a medida que vaya incrementándose el valor de del bucle N iremos obteniendo los valores de color que nos interesan. Mira los cambios del código siguiente y lo comprenderás:

```
5 PAPER 0: BORDER 0: CLS
10 FOR n=1 TO 200: PRINT AT RND*19+1,RND*29+1;INK (n/50)+3;"":NEXT
   n
19 BEEP 1,20
20 PRINT "the end!"
```

11,8 sg ¡Vayaaaa!! Esto va pintando bien. Ahora vamos a indicarle al intérprete ***ZXBASIC*** que tanto los valores aleatorios que estamos usando para coordenadas como el valor ***N/50*** que obtenemos para el color de los cuerpos celestes del firmamento (y que puede ser a veces un valor decimal), son del tipo entero. Así comparamos la diferencia de usar a no usar esta función que descarta la parte decimal de un número. Para ésto, usaremos la función ***INT*** combinándola con la función ***RND*** ó directamente delante del valor que queremos (***N/50***). Veamos en qué resulta nuestra ocurrencia...

```
5 PAPER 0: BORDER 0: CLS
10 FOR n=1 TO 200: PRINT AT INT(RND*19)+1,INT(RND*29)+1;INK
   INT(n/50)+3;"": NEXT n
19 BEEP 1,20
```

```
20 PRINT "the end!"
```

11,2 sg ¡¡Buenooooo!! No está mal. En cuatro pasadas con nuestro paño mágico y algo de insistencia hemos logrado reducir el tiempo de ejecución de nuestra rutina desde los 15,5 sg iniciales a los actuales 11,2 sg. Si crees que no es demasiado, veamos con una sencilla regla de tres que nos muestra de qué mejora porcentual estamos hablando:

100__________**15.5** s (tiempo inicial)

x___________**11.2** s (tiempo óptimo)

x = 1120/15.5= 72.6% >>> 100 - 72.6 = 27,4 %

Sinceramente, supongo que igual que yo, pensarás que un mejora del 27,4 % es ya una cifra a tener en cuenta, sin embargo, finalmente podríamos incluso optar por decisiones más drásticas para dar una vuelta de tuerca al código aunque a veces pueda ser a costa de perder algo de vistosidad. Al fin y al cabo, tal vez esa pérdida no suponga un trauma severo para nuestro programa, pero eso es algo que debemos valorar de forma muy meditada para intentar conseguir un resultado lo más equilibrado posible, en este caso, entre velocidad y belleza. Veamos pues un ejemplo para que veas de que hablo cuando me refiero a este dilema universal (velocidad-belleza).

Vamos a omitir directamente el espectro/rango de color aleatorio de los cuerpos celestes y vamos a suponer que el color de los mismos será siempre blanco, eso sí, con un atributo de brillo variable, de manera que la mitad de los cuerpos del firmamento tendrán atributo de brillo CERO (sin brillo) y la otra mitad brillo UNO (con brillo). No sé si ganaremos algo de tiempo, conozcamos pues la sentencia del crono en este caso:

```
5 PAPER 0: BORDER 0: ink 7: CLS
10 FOR n=1 TO 200: PRINT AT INT(RND*19)+1,INT(RND*29)+1; BRIGHT
   INT(n/200);"": NEXT n
19 BEEP 1,20
20 PRINT "the end!"
```

11,1 sg. Bueno, esto sí es casi insignificante. Aunque tal vez podríamos utilizar una fórmula de optimización a la que podríamos denominar ***desdoblamiento*** y que seguramente nos reportaría un mejor tiempo a costa de algo de memoria. Esta técnica del desdoblamiento consiste básicamente en desdoblar el bucle de 200 en dos de 100, de manera que el primero imprimiría los caracteres con un atributo de brillo 0 (apagado) por ejemplo, y el segundo bucle con el atributo de brillo a 1 (encendido). Pero... ¿Qué ventajas ofrece este sistema de desdoblamiento si hay que escribir más código?

Pues la ventaja es que podemos dejar fuera de los bucles (antes de la orden ***FOR***) los comandos ***BRIGHT***

0/BRIGHT 1 con la consiguiente ventaja en la velocidad.

Seguro que mirando el código lo terminas de entender:

```
5 PAPER 0: BORDER 0: INK 7: CLS: BRIGHT 0
10 FOR n=1 TO 100: PRINT AT INT (RND*19)+1,INT
   (RND*29)+1;"": NEXT n
14 BRIGHT 1
15 FOR n=1 TO 100: PRINT AT INT (RND*19)+1,INT
   (RND*29)+1;"": NEXT n
19 BEEP 1,20
20 PRINT "the end!"
```

10,0 sg ¡¡Voilà!! Ni el crono miente ni mis augurios eran fruto de la sugestión sino del ¡razonamiento lógico computacional! ¡Uyy!! ¡Qué chulo ha quedado eso!!. En resumen, con respecto a la versión anterior hemos añadido algo más de código (gasto de memoria) pero a cambio hemos podido compactar un poco más el código que se repite dentro del bucle y obteniendo un beneficio interesante de velocidad, además, ¿No se te ocurre alguna manera de reducir un poquito más el código? ¿Por qué no compactarlo un poco más metiendo los dos bucles en una única línea múltiple? Tal vez funcione y además ganemos algo de tiempo. Veamos...

```
5 PAPER 0: BORDER 0: INK 7: CLS: BRIGHT 0
10 FOR n=1 TO 100: PRINT AT INT (RND*19)+1,INT (RND*29)+1;"": NEXT n:
```

```
BRIGHT 1: FOR n=1 TO 100: PRINT AT INT (RND*19)+1,INT
(RND*29)+1;"": NEXT n
19 BEEP 1,20
20 PRINT "the end!"
```

9,6 sg Pues sí, eso de meter varios comandos en una única línea parece resultar siempre bastante efectivo. Ahora que conocemos esta técnica de ***desdoblamiento*** podría emplearse perfectamente para dibujar cuerpos celestes en varios colores o en otras muchas situaciones.

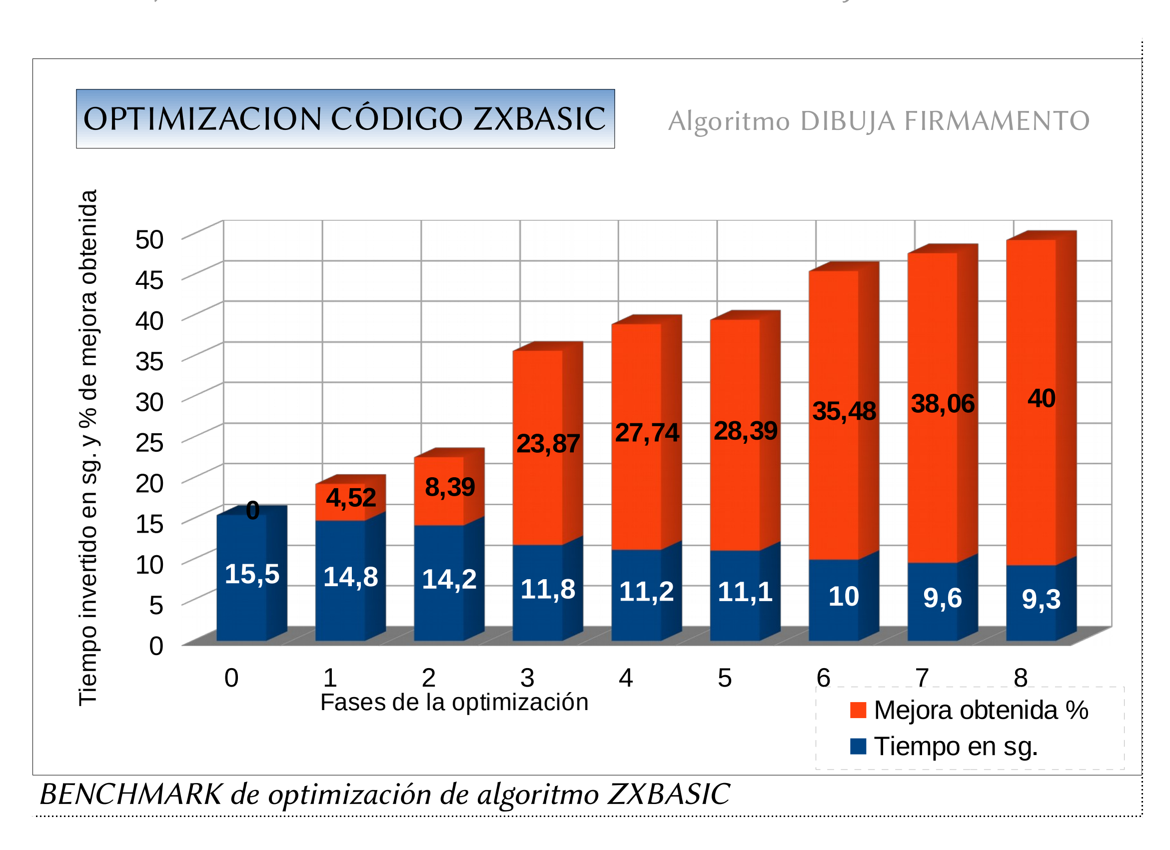
Y ya para acabar, vamos a probar con algo mucho más drástico como es omitir directamente el atributo de color/brillo de los cuerpos celestes y dejarlos con valor único establecido con anterioridad al inicio del bucle:

```
5 PAPER 0: BORDER 0: INK 7: BRIGHT 1:CLS
10 FOR n=1 TO 200: PRINT AT INT(RND*19)+1,INT(RND*29)+1;"": NEXT n
19 BEEP 1,20
20 PRINT "the end!"
```

9,3 sg Aunque solo hemos rebajado unas pocas décimas, esta es la opción por la que me decanté finalmente para nuestro juego tipo, entre otras cosas, porque el hecho de asignar idéntico atributo a todos los cuerpos celestes del firmamento, luego nos facilitar el método empleado para que nuestro protagonista pueda pasar sobre los cuerpos celestes sin borrarlos haciendo uso del comando ATTRIB

para leer los atributos de las diferentes posiciones de la pantalla. Todo esto está explicado en la sección de *Análisis Profundo del Código línea a línea*.

Llegados a este punto, veamos como ha mejorado la velocidad de nuestro sencillo algoritmo "dibuja firmamento" de una forma más visual. En la siguiente gráfica puede apreciarse la evolución de las distintas fases de la optimización (desde la cero o estado inicial hasta la 8), los tiempos de procesamiento se muestran en segundos (color azul) mientras que las mejoras obtenidas en porcentaje en cada fase de la optimización (con respecto al estado inicial) se muestran en las columnas rojas.



BENCHMARK de optimización de algoritmo ZXBASIC

Aunque nunca se sabe que gratas sorpresas puede depararnos la técnica del paño abrillantador, tras una revisión exhaustiva de nuestro código es bastante probable que consigamos mejorar la *velocidad* y/o ahorremos *memoria*, en algunas ocasiones tal vez debamos decidirnos por una de las dos opciones, pero lo mejor de todo es que estas mejoras y optimizaciones que consigamos sobre nuestro programa se beneficiarán aún más al pasar nuestro trabajo por el compilador ***MCODER3*** (o el que hayamos elegido en su lugar). Piensa que todo este estrujamiento del código, a veces casi obsesivo, no solo llevará a nuestro desarrollo en ***ZXBASIC*** al límite del rendimiento de la máquina sino que puede acabar marcando la diferencia en cuanto a la calidad técnica de nuestro juego.

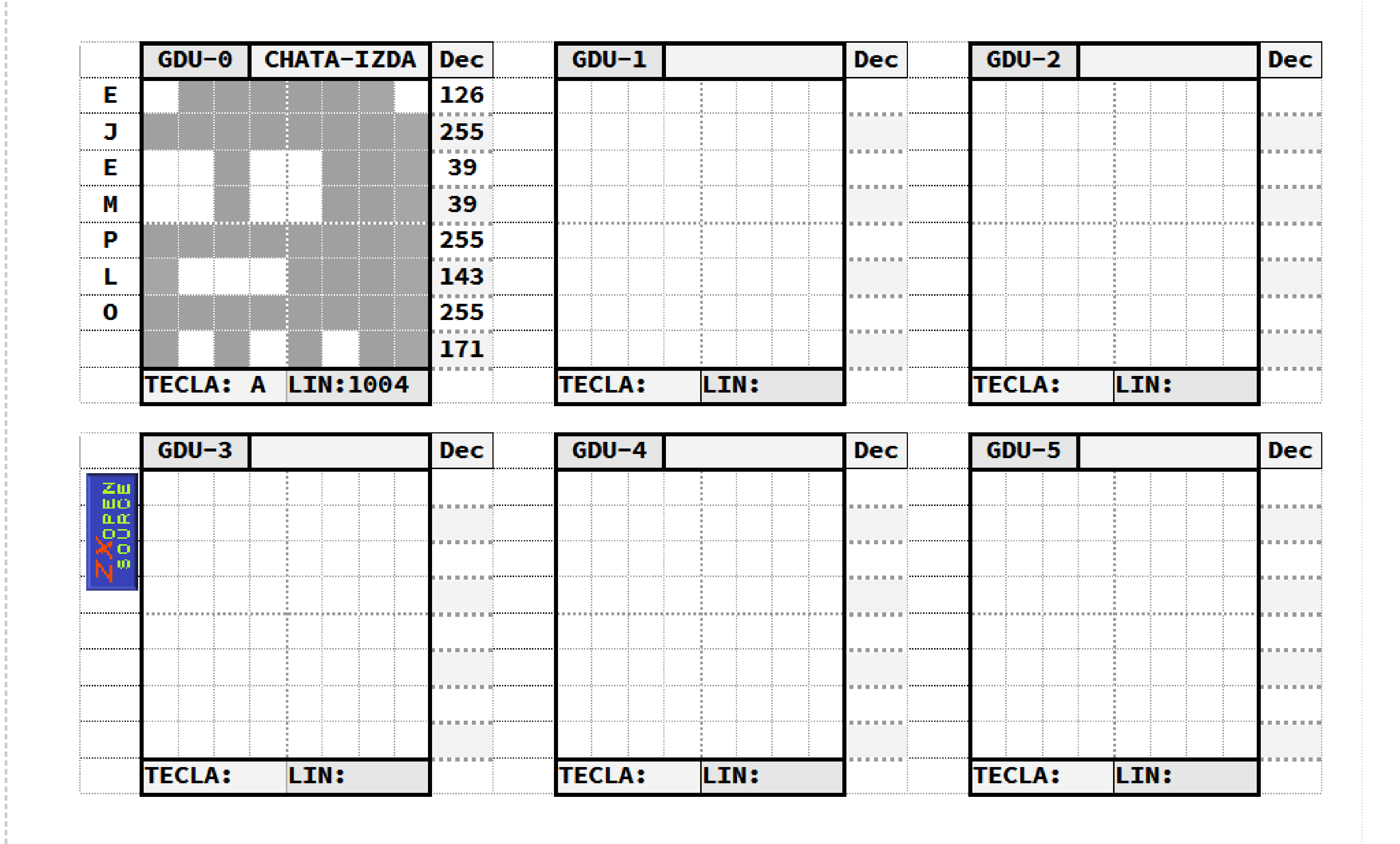
Por ello, aunque personalmente recomendaría hacer todas las pruebas necesarias por costumbre, es importante incidir y centrarse sobre todo en los procesos de mayor carga y que suelen ser los bucles.

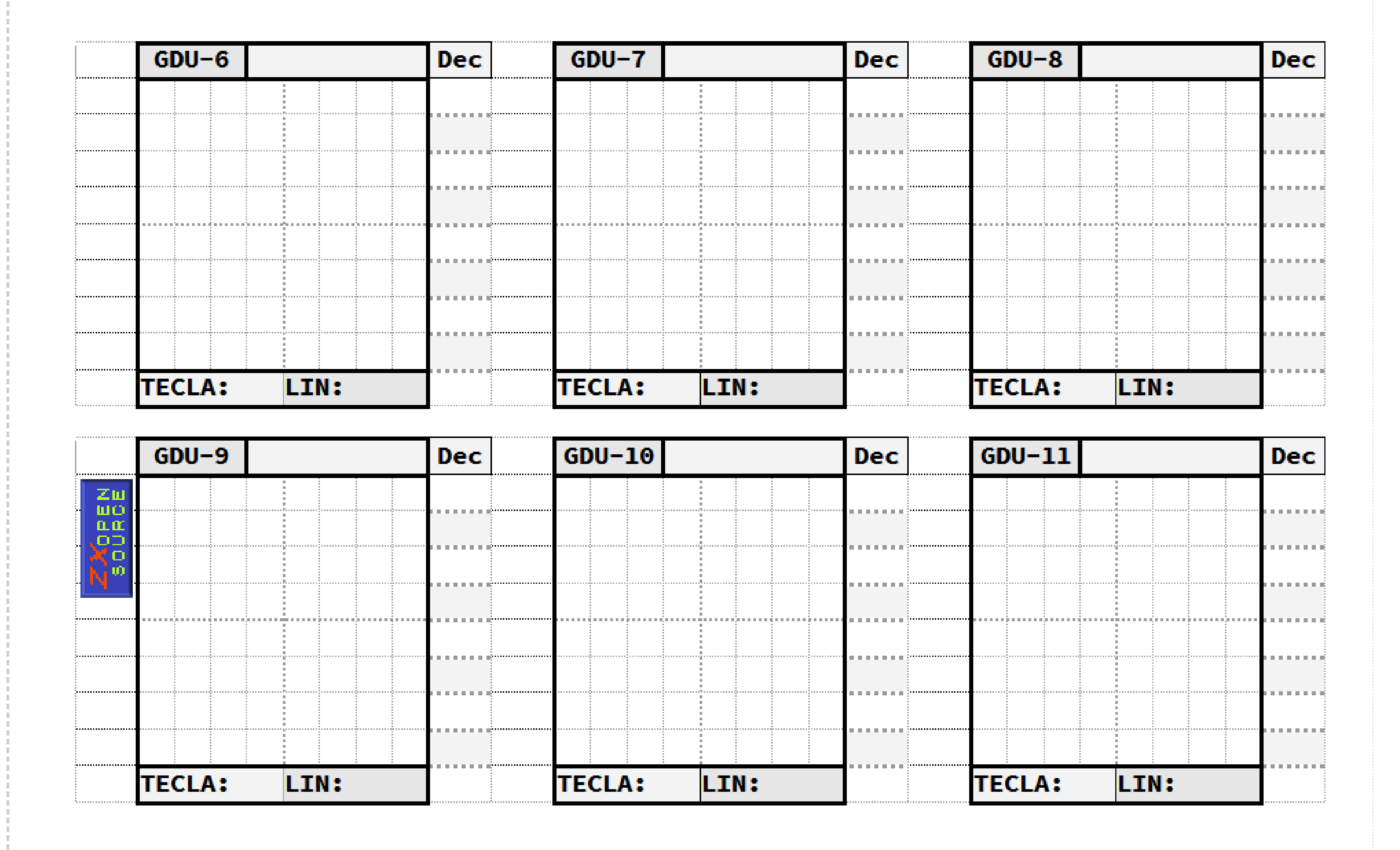


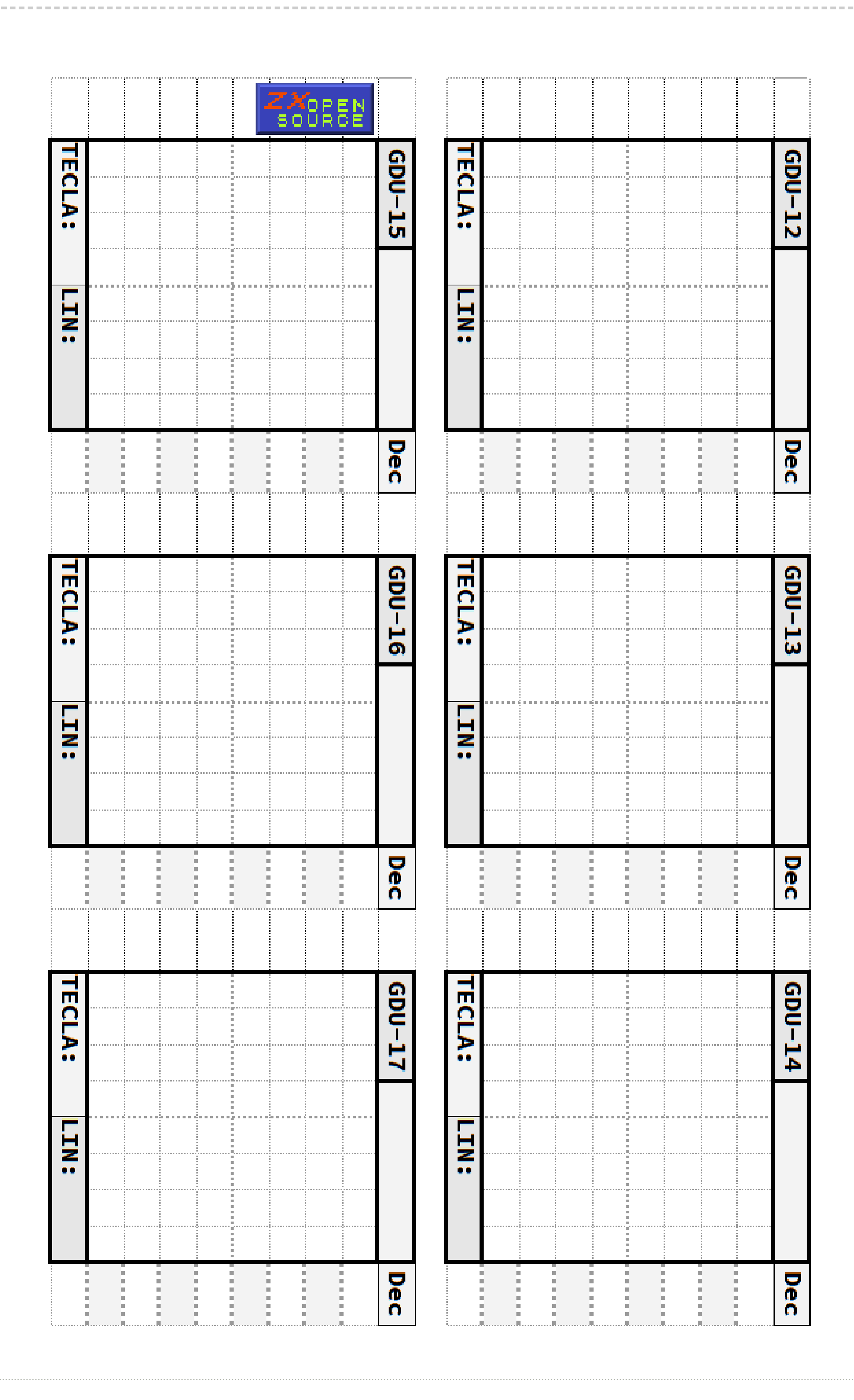
En la sección de ANEXOS voy a incluir algunas cositas interesantes que facilitarán nuestro trabajo. Esto es lo que he preparado:

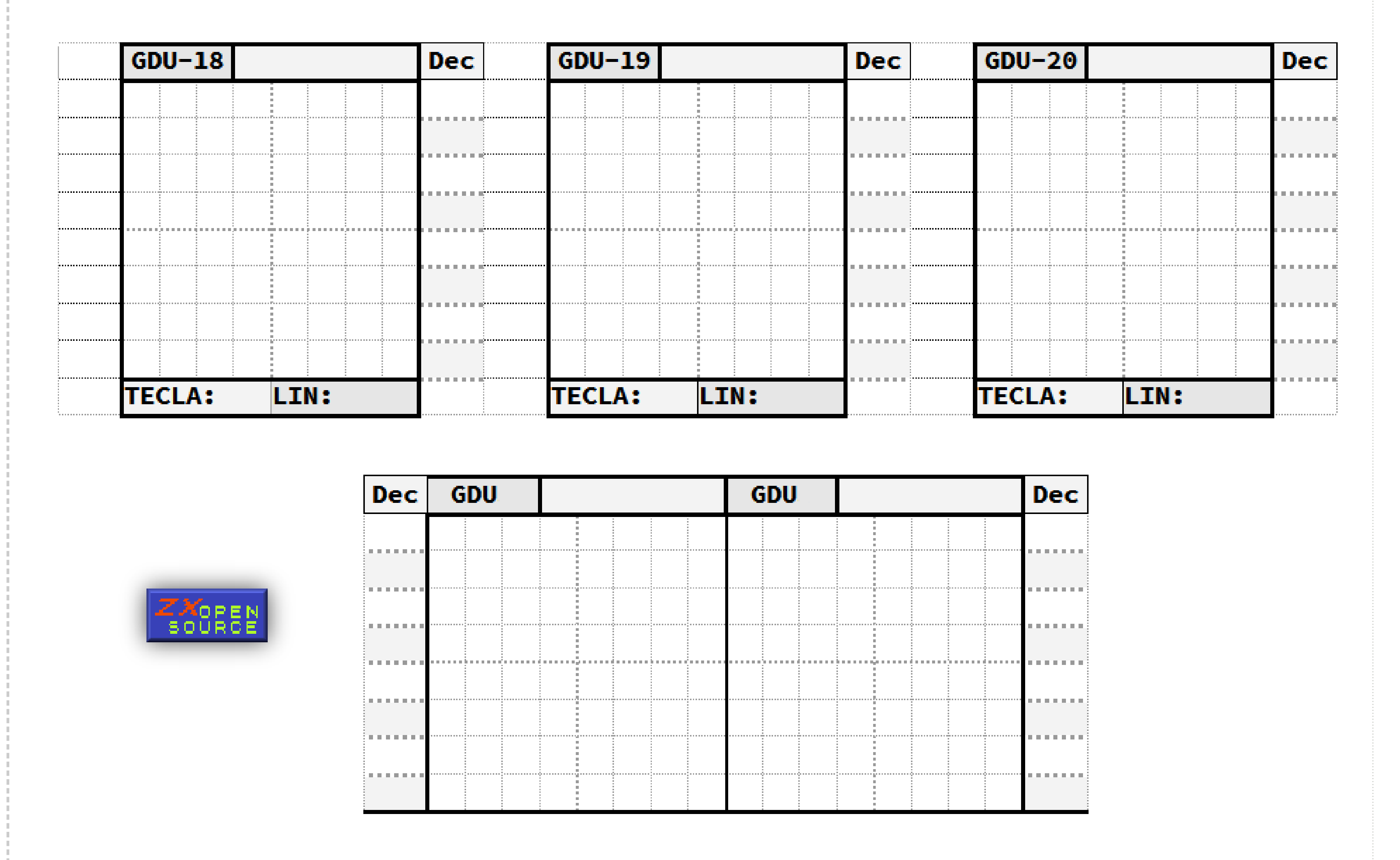
- *Plantilla de pantalla en baja resolución.*
- *Plantilla de pantalla en alta resolución.*
- *Plantillas de sprites GDUs de 8x8 píxeles.*
- *Directorio de enlaces útiles.*
- *Glosario retromaníaco de ZX-Spectrum.*

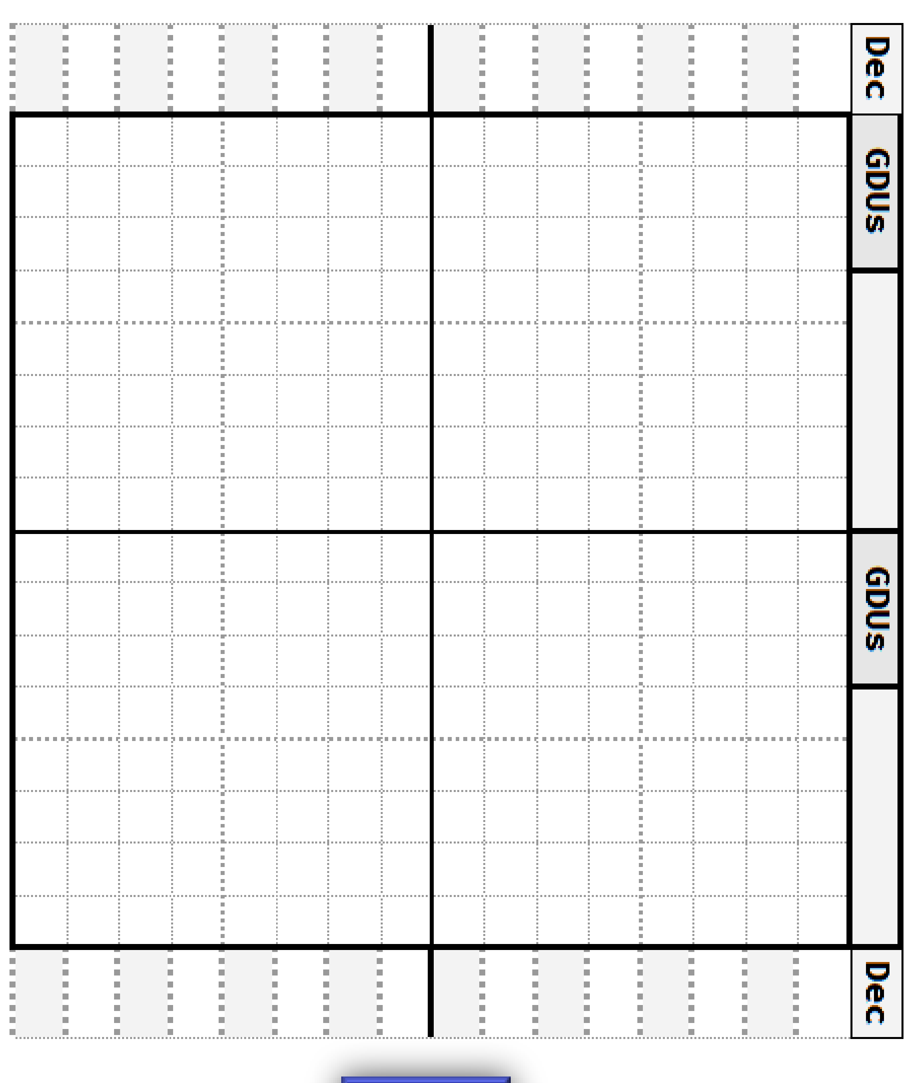

0,0
0,31
fila 0 (coordenada Y)
columna 0 (coordenada X)
columna 31 (coordenada X)
PRINT AT Y,X;"HOLA MUNDO"
PRINT #1;"IMPRIME ABAJO"
PRINT #2;"IMPRIME MAS ABAJO"
21,0
fila 21 (coordenada Y)
21,31
fila #1
fila #2


GDU-0
CHATA-IZDA
Dec
E
J
E
M
P
L
O
126
255
39
39
255
143
255
171
TECLA: A
LIN:1004
GDU-1
Dec
TECLA:
LIN:
GDU-2
Dec
TECLA:
LIN:
ZX OPEN SOURCE
GDU-3
Dec
TECLA:
LIN:
GDU-4
Dec
TECLA:
LIN:
GDU-5
Dec
TECLA:
LIN:

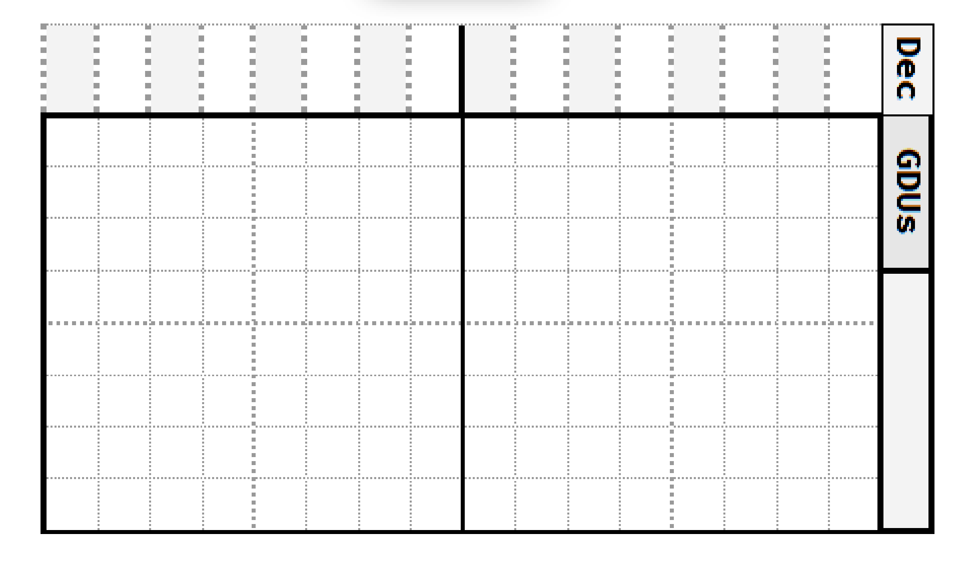

GDU-6
Dec
TECLA:
LIN:
GDU-7
Dec
TECLA:
LIN:
GDU-8
Dec
TECLA:
LIN:
ZX OPEN SOURCE
GDU-9
Dec
TECLA:
LIN:
GDU-10
Dec
TECLA:
LIN:
GDU-11
Dec
TECLA:
LIN:


ZX OPEN SOURCE
GDU-15
TECLA:
LIN:
Dec
GDU-16
TECLA:
LIN:
Dec
GDU-17
TECLA:
LIN:
Dec
GDU-12
TECLA:
LIN:
Dec
GDU-13
TECLA:
LIN:
Dec
GDU-14
TECLA:
LIN:
Dec


GDU-18
Dec
TECLA:
LIN:
GDU-19
Dec
TECLA:
LIN:
GDU-20
Dec
TECLA:
LIN:
Dec
GDU
GDU
Dec
ZX OPEN SOURCE


Dec
GDUs
GDUs
Dec


ZX OPEN
SOURCE


Dec
GDUs

Dec	GDUs		GDUs		Dec